VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA

EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

Vývoj Rich-Client aplikace pro průmyslovou společnost

Development of Rich-Client Application for Industrial Company

Student: Miroslav Zapletal

Vedoucí bakalářské práce: Ing. Vítězslav Novák, PhD.

Ostrava 2011

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

Zadání bakalářské práce

Student: **Miroslav Zapletal**

Studijní program: B6209 Systémové inženýrství a informatika

Studijní obor: 6209R001 Aplikovaná informatika

Téma: Vývoj Rich-Client aplikace pro průmyslovou společnost
Development of Rich-Client Application for Industrial Company

Zásady pro vypracování:

1. Úvod
2. Popis technologií použitých při vývoji aplikace
3. Analýza stávajícího stavu
4. Implementace nové funkcionality a zhodnocení
5. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků bakalářské práce
Přílohy

Seznam doporučené odborné literatury:

BÖCK, H. *The Definitive Guide to NetBeans Platform.* 1st ed. Berkeley: Apress, 2009. 450 s. ISBN 978-1-4302-2417-4.
PETRI, J. *NetBeans Platform 6.9 Developer's Guide.* 1st ed. Birmingham: Packt Publishing, 2010. 288 s. ISBN 978-1-849511-76-6.
ZAHKOUR, S.; HOMMEL, S.; ROYAL, J.; RABINOVITCH, I.; RISSER, T.; HOEBER, M. *Java 6 Výukový kurz.* 1. vyd. Brno: Computer Press, 2007. 536 s. ISBN 978-80-251-1575-6.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Vítězslav Novák, Ph.D.**

Datum zadání: 26.11.2010
Datum odevzdání: 11.05.2011

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Ekonomická fakulta 1 OSTRAVA

Ing. Jan Ministr, Ph.D.
vedoucí katedry

prof. Dr. Ing. Dana Dluhošová
děkanka fakulty

„Místopřísežně prohlašuji, že jsem celou práci, včetně všech příloh, vypracoval samostatně.“

V Ostravě dne 9. května 2011 ……

Miroslav Zapletal

Rád bych poděkoval panu Ing. Vítězslavu Novákovi, PhD. za ochotu a rady, které přispěly k vypracování této práce. Dále velmi děkuji celému týmu ERDC ve společnosti ON Semiconductor za poskytnutí výborného zázemí.

Obsah

Úvod

V dnešní době nemůže moderní společnost úspěšně obstát na trhu bez dobré IT infrastruktury. Moderní informační systém může výrazně zjednodušit výrobu, rozhodování, plánování a řízení celé společnosti.

Nezáleží na tom, zda je řeč o ERP systémech, či výrobních systémech. Je velice efektivní automatizovat procesy ve společnostech a počítače jsou v tomto směru úžasným pomocníkem. Existují obory, které by nemohly existovat bez počítačem řízených procesů, například výroba polovodičů.

Vývoj takových systémů ovšem představuje velice nákladnou činnost. Do projektu je zapotřebí vložit mnoho úsilí a o to více peněz, aby takový systém, v případě umístění na trhu, byl konkurenceschopný. Je třeba mít ustanovený kvalitní analytický a programátorský tým, který je velice flexibilní a dokáže využívat nejmodernější technologie, které v oblasti existují.

Analogicky k tomu, jaká je poptávka po informačních systémech, je také veliká poptávka po vývojářích takových systémů. Současnému trendu odpovídá také neustálý růst mezd v IT oboru, díky němuž se obor stává velice lukrativní. Nemohu ale v této souvislosti hovořit globálně, jelikož ne všechny země jsou stejně technicky zaměřené.

V neposlední řadě nemůžeme také pominout fakt, že na školách se vyučují všechny potřebné předměty pro vývoj informačních systémů. Vyučují se databáze, vyučuje se programování, vyučuje se také modelování informačních systémů. Všechny tyto znalosti můžeme velice snadno uvést do praxe a zúročit investice vložené do studia.

Ke zpracování své práce jsem využil letní stáže ve společnosti ON Semiconductor, která se zabývá výrobou polovodičových součástek. Nastoupil jsem v Rožnově pod Radhoštěm do evropského vývojového centra s cílem, dozvědět se něco o moderních technologiích využívaných při tvorbě rozsáhlých aplikací a aktivně se zapojit do jejich vývoje. Pracoval jsem na aplikaci, která zobrazuje jednotlivé čipy na křemíkových deskách.

Cílem mé práce není vytvoření systému pro výrobu polovodičů. To je práce na dlouhé měsíce pro zkušený vývojový tým. Mým záměrem je představit technologie, které se na tomto poli využívají, jak vypadají, jak se s nimi pracuje a co dělají. Dále v práci analyzuji prostředí pro běh takových aplikací, datová úložiště včetně jejich

zabezpečení. Součástí práce je i popsání implementace nových funkcionalit, které jsem během své stáže řešil.

1 Popis technologií použitých při vývoji aplikace

1.1 Java

Technologie Java je programovací jazyk a platforma. Programovací jazyk Java je jazyk vyšší úrovně a lze o něm říci, že je objektově orientovaný, nezávislý na architektuře, robustní, zabezpečený. Určitě bychom našli mnoho dalších výrazů, jak tento jazyk charakterizovat. Programování v jazyce Java obnáší psaní zdrojových kódů, které se uchovávají v textové podobě. Tento kód následně kompilátor zkompiluje do souborů *.class*. Tyto soubory ovšem neobsahují nativní kód pro daný procesor, ale tzv. bytecode virtuálního stroje Javy. Spuštění aplikace potom probíhá s vytvořenou instancí JVM[1], která překládá bytecode do kódu strojového. JVM je k dispozici pro mnoho různých operačních systémů. Výše uvedený mechanismus nám proto umožňuje psát programy pro tyto systémy prakticky totožným způsobem. [4]

Platforma Java je softwarová platforma obsahující dvě součásti – Java Virtual Machine a API. Java Virtual Machine slouží k výše popsanému účelu. API je sbírka hotových komponent pro použití při programování. Tyto komponenty jsou roztřízeny do tzv. baličů (packages). [4]

1.1.1 Swing

Rozhraní Swing je součástí Javy od verze 1.2 a obsahuje komponenty, které programátorovi umožňují tvorbu GUI[2]. Celé rozhraní by bylo zbytečné bez implementace základních funkcí, jako jsou obsluha událostí, správce oken, přizpůsobitelné vykreslování či funkce *drag-and-drop*. Swing nabízí také podporu pro operace *undo* (funkce zpět), internacionalizaci, případně možnost změny vzhledu grafických komponent, včetně vytváření motivů vlastních. [4]

1.1.2 Java EE

Platforma Java Enterprise Edition je nadstavbou klasické verze Javy – Java Standard Edition. Je předurčena pro vývoj rozsáhlých, škálovatelných, spolehlivých a zabezpečených síťových aplikací. Tyto aplikace jsou označovány jako „enterprise“, protože jsou často nasazovány do velkých organizací. Nelze ovšem tvrdit, že jsou

[1] Java Virtual Machine
[2] Grafické uživatelské rozhraní

výhradně zaměřené pro velké společnosti. Výhody Javy EE může využít i individuální programátor či malá organizace. [16]

Enterprise aplikace je rozdělena do několika vrstev (anglicky tiers): [16]

- **Klientská vrstva (client tier)** – tato vrstva definuje klienty aplikace, mohou to být webové prohlížeče nebo jiné aplikace
- **Webová vrstva (web tier)** – v této vrstvě se nachází komponenty zabezpečující interakci mezi klienty a serverem a využívají se zde technologie, jako jsou Java Servlets, Java Server Pages, Java Server Faces, Expression Language apod.
- **Business vrstva (business tier)** – tato vrstva je jádrem dobře navržené aplikace a při použití technologií jako Java Persistence API nebo JAX-WS se v ní vykonávají úlohy, pro které je aplikace původně určena (obsahuje její logiku)
- **Vrstva EIS (enterprise information system tier)** – často funguje na více než jednom serveru a nachází se zde databázové servery, ERP systémy a různé datové zdroje

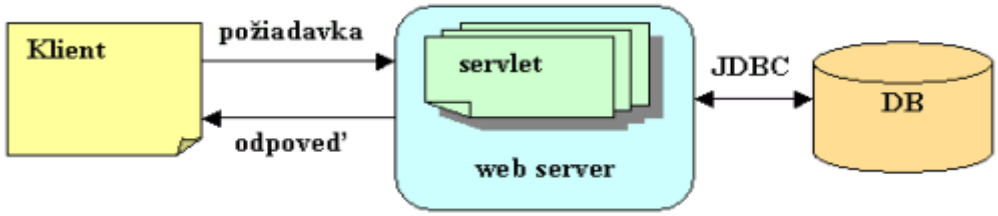
1.1.3 Java Servlets

Servlety jsou programové komponenty běžící na straně serveru a jsou součástí Javy EE. K jejich využívání je zapotřebí minimálně aplikační server podporující tuto technologii, který bude mít přístup k JVM. Servlety se nejčastěji používají nad protokolem HTTP, přičemž zpracovávají klientské požadavky. Nemají vlastní GUI, avšak mohou generovat výstup viditelný v internetových prohlížečích. [7]

Nejsou vhodné pro použití jako klasické webové stránky. Výstup se definuje čistě programově přes výstupní proudy. Proto se tento postup nedoporučuje. Servlet se ovšem velice dobře hodí do pozice prostředníka mezi aplikací a standardními webovými stránkami.

Servlet funguje na principu požadavku klienta a odpovědi serveru. Klient vytvoří požadavek a pošle jej na server. Ten požadavek přepošle na příslušný servlet, který jej zpracuje. Odpověď se předá zpět serveru, který ji pošle příslušnému klientovi. [7]

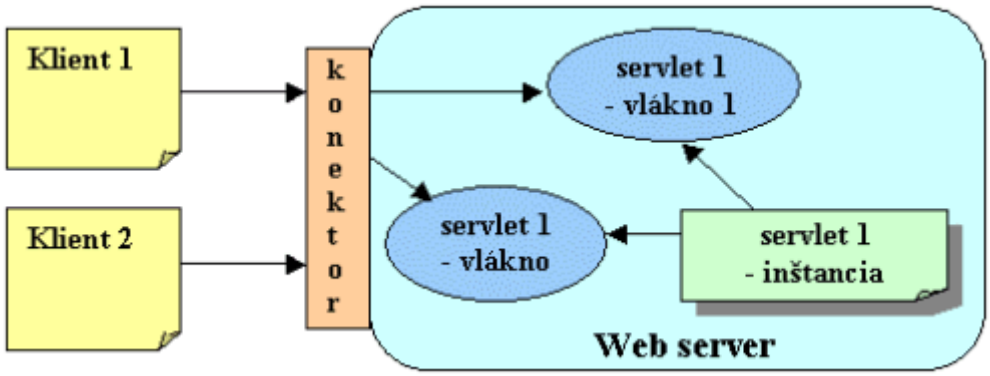
Obrázek 1.1



Zdroj: [7]

Servlet existuje na serveru pouze v jediné instanci (jde o singleton). Při vyřizování více požadavků najednou je k servletu přistupováno více vlákny – pro každý požadavek jedno vlákno. Tento princip snižuje využití systémových zdrojů. Je ale důležité mít tento fakt neustále na mysli a programování servletu tomu přizpůsobit. [7]

Obrázek 1.2



Zdroj: [7]

1.2 Rich-Client

Rich-Client je rozsáhlá aplikace zpracovávající data na straně klienta (pokud se jedná o architekturu klient-server) a poskytuje grafické uživatelské rozhraní (GUI). Jedná se o aplikace, které se dají rozšiřovat pomocí zásuvných modulů a tyto moduly řeší širokou škálu problémů, které spolu kolikrát ani nemusí přímo souviset. [1]

Rich-Client aplikaci poskytuje nějaký základní rámec (framework) infrastrukturu, na které následně programátor logicky aplikaci vyvíjí v jednotlivých modulech. Takový vývoj může dojít i do situace, kdy se několik modulů tváří jako modul jeden (např. při spolupráci modulů vyvíjenými různými výrobci). Aplikace může tak být seskládána z různých modulů podle přání zákazníka. Taková aplikace je také velice snadno aktualizována z internetu, případně se z internetu může přímo spouštět (např. pomocí technologie Java Web Start). [1]

Přehled jednotlivých vlastností Rich-Client aplikace: [1]

- Poskytuje grafické uživatelské rozhraní
- Je vyvíjena modulárně
- Je nezávislá na platformě
- Snadno se aktualizuje
- Zpracovává data na straně klienta (pokud se jedná o architekturu klient-server)

V současné době se využívají především dvě nejrozšířenější Rich-Client platformy – Eclipse RCP a NetBeans RCP.

1.3 Platforma NetBeans

1.3.1 Platforma NetBeans

Platforma NetBeans umožňuje vyvíjet rozsáhlé modulární (rich-client) aplikace využívající Java Swing. Poskytuje prostředí pro životní cyklus takové aplikace a spravuje tak „život“ jednotlivých modulů. Vývoj takových aplikací je rozčleněn do jednotlivých, na sobě nezávislých (pokud závislost nevyžadujeme) modulů, které podstatně zpřehledňují celkový kód. Modul může být prakticky cokoli od grafické komponenty, přes uživatelskou akci, knihovnu, až po službu, která je využívána dalšími moduly. Tento framework již obsahuje API pro uživatelské akce, klávesové zkratky, aktualizace, management oken a ostatní prvky, které má mnoho aplikací společných. Odpadá tím programování těchto komponent od základu, což je při psaní běžné swingové aplikace nezbytné. [15]

Dále bych zmínil jednu obrovskou výhodu platformy – některé části frameworku (závislosti mezi moduly, nabídky, pozice komponent atd.) se konfigurují deklarativně. Platforma má vlastní API pro načítání těchto nastavení z XML souborů uvnitř jednotlivých modulů. Velice elegantně se tak vyřešilo oddělení GUI od vlastního kódu aplikace, což umožňuje daleko větší flexibilitu při vývoji. [1]

Dokonce i jednotlivé moduly jsou deklarativně popsány a platformou automaticky načítány. Odpadá tím zavedení vazeb mezi kódem a aplikací. Tyto vazby nejsou ani mezi jednotlivými moduly, takže celá aplikace je velice dynamicky rozšiřitelná a může se na přání zákazníka z jednotlivých modulů sestavit přímo „na míru“. [1]

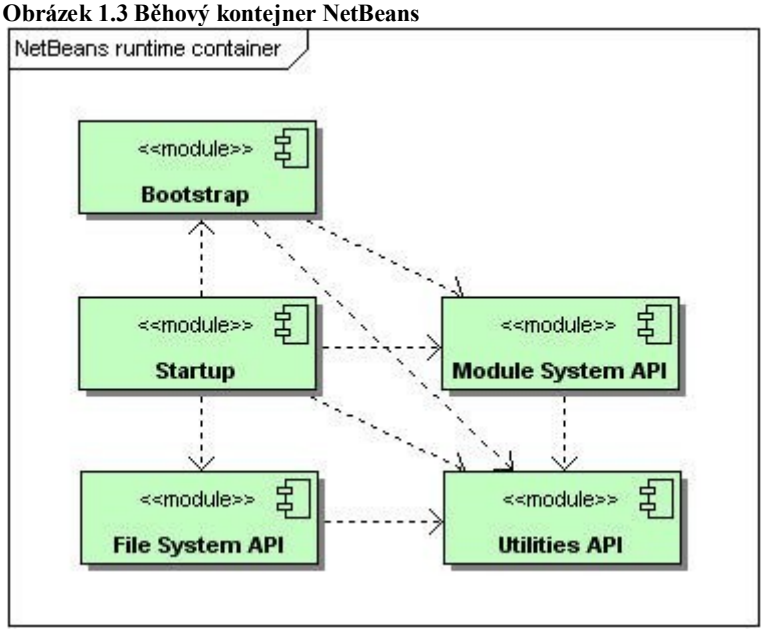
1.3.2 Běhový kontejner NetBeans

Běhový kontejner je základem celé modulární architektury. Každý modul musí mít svůj životní cyklus. Modulární systém NetBeans spravuje načítání a konfiguraci takových modulů. To také zahrnuje vykonání instalačního kódu těchto modulů (pokud nějaký existuje) a také správné uvolnění jednotlivých modulů při ukončení aplikace. [3]

Běhový kontejner tvoří minimální skupinu modulů, které platforma potřebuje pro svůj běh. Skládá se z pěti následujících modulů: [1]

- Bootstrap
- Startup
- Module System
- File System
- Utilities

Obrázek 1.3 Běhový kontejner NetBeans



Zdroj: http://platform.netbeans.org/images/tutorials/runtime-container/runtime-container.jpg

Obr. 1.3 zobrazuje vztahy mezi jednotlivými moduly běhového kontejneru. Jako první je spuštěn modul *Bootstrap*. Je zodpovědný za připravení zavaděče jednotlivých tříd, ten následně zavolá modul *Startup*, který startuje aplikaci a následně inicializuje modulový systém (ten odpovídá za správu jednotlivých modulů) a souborový systém (zpřístupní virtuální datový systém). Modul *Utilities* poskytuje základní nástroje např. pro komunikaci mezi moduly. [1]

1.3.3 Modulový systém NetBeans

Modulový systém obstarává veškerou správu modulů v aplikaci a s tím spojené úkony – instalování, aktivace, deaktivace apod. Jinými slovy se dá říct, že modulový systém obstarává životní cyklus jednotlivých modulů. [1] [3]

Při navrhování modulového systému se nejvíce vycházelo z technologií Javy. Velice blízký koncept představuje klasický JAR archiv. Programátor může vytvořit aplikaci, případně knihovnu, tu následně zkompilovat a zabalit do JAR archivu. V případě potřeby může do archivu přidat libovolný počet zdrojů využívaných v aplikaci. Celý JAR soubor následně popíše v souboru manifest.mf, který se v archivu také nachází. [3]

Modulový systém funguje velice podobně. Využívá se JAR archiv, případně NBM archiv (prakticky je totéž co JAR, používá se jiná koncovka pro lepší odlišení). Tento archiv obsahuje i veškeré zdroje potřebné pro fungování modulu. V manifestu se modul popíše včetně jeho jedinečného identifikátoru. [3]

Manifest ovšem není jediný „popisovací“ soubor v NBM archivu. Stále je nutné definovat, jak integrovat modul do platformy. To je uloženo v souboru layer.xml. [1]

1.3.4 Struktura modulu

Každý modul musí splňovat přesné požadavky, aby mohl být do platformy integrován. Ten nejdůležitější je ve své podstatě i nejjednodušší – modul musí mít vlastní identitu.

Modul obsahuje tyto náležitosti:

- Soubor manifest.mf
- Soubor layer.xml
- Soubory .class
- Zdroje (ikony, soubory s nastavením a lokalizací apod.)

Povinný je přitom pouze manifest, ostatní závisí na účelu použití modulu. Modul také obsahuje XML soubor, který má stejný název jako modul (včetně názvu balíčku). Tento soubor je načítán jako první a oznamuje platformě existenci modulu. [1]

1.3.5 Soubor manifest.mf

Manifest je soubor, který je obsažen v každém modulu platformy. Obsahuje informace pro popis modulu a jeho prostředí. Jeho jediný povinný atribut je *OpenIDE-Module*, který jedinečně identifikuje daný modul. Konvence radí, že by se měl používat celý název modulu včetně názvu balíčku, pro zabránění konfliktů s moduly od jiných výrobců. [1]

Tento modul ovšem obsahuje i jiné atributy, kterých je poměrně mnoho. Mimo jiné obsahuje deklaraci závislostí, název inicializátoru modulu (třída vykonávající inicializační akce v průběhu životního cyklu modulu) a jiné atributy. [1]

Ukázka souboru manifest.mf:

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.8.0
Created-By: 1.6.0_18-b07 (Sun Microsystems Inc.)
OpenIDE-Module-Public-Packages: org.openide.*
OpenIDE-Module-Module-Dependencies: org.netbeans.api.progress/1
> 1.6,
  org.openide.awt > 6.2, org.openide.util > 8.0,
org.openide.util.look
 up > 8.0
OpenIDE-Module-Java-Dependencies: Java > 1.6
OpenIDE-Module-Implementation-Version: 201007282301
```

```
AutoUpdate-Show-In-Client: false
OpenIDE-Module: org.openide.dialogs
OpenIDE-Module-Specification-Version: 7.15.1
OpenIDE-Module-Localizing-Bundle: org/openide/Bundle.properties
AutoUpdate-Essential-Module: true
OpenIDE-Module-Requires: org.openide.modules.ModuleFormat1
```

1.3.6 Soubor layer.xml

Běhový kontejner NetBeans musí vědět, jak jednotlivé moduly do platformy integrovat. K tomu slouží soubor layer.xml, který se nachází uvnitř modulů. Tento soubor obsahuje definice prakticky všeho, co je zapotřebí k integraci modulu. Integrace tudíž není otázkou programového kódu, nýbrž je nastavována deklarativně. [1]

Všechny soubory layer.xml jsou sjednoceny ve virtuálním souborovém systému. Podle něj následně platforma generuje příslušná okna, panely nástrojů, nabídky, uživatelské akce apod. Proto není problém při vývoji nového modulu přidat například novou nabídku do menu, aniž bych dělal úpravy ve zdrojovém kódu aplikace (třeba při vývoji zásuvného modulu pro NetBeans IDE). [3]

1.3.7 Typy modulů

Existují celkem tři typy modulů. Rozdíl mezi nimi spočívá ve způsobu načítání platformou. Vhodným nastavením lze docílit rychlejšího startu a chodu aplikace. Modul typu **Regular** je nejběžnější. Zavádí se při startu aplikace, který je ovšem o tuto inicializaci delší. Proto je důležité neprovádět mnoho úkolu při načtení modulu, ale odsunout je na pozdější dobu (pokud je to možné). Dalším typem je **Autoload** a jak již název napovídá, jde o modul, který se načítá, až pokud ho jiný modul vyžaduje. Často se využívá v modulech, které zaobalují externí knihovny. Posledním typem je typ **Eager**, který také řeší otázku zkrácení času spouštění aplikace. Takový modul je načten až tehdy, kdy jsou splněny všechny jeho závislosti. Není totiž nutné načítat modul, když nejsou dostupné jiné moduly, na kterých je závislý. [1]

1.3.8 Lookup

Lookup je jednou ze základních komponent platformy NetBeans. Umožňuje komunikaci mezi jednotlivými moduly. Funguje na principu mapy, do které se ukládají třídy a instance objektů. Pomocí třídy objektu získáme příslušnou instanci. [15]

To přináší obrovskou výhodu v typové bezpečnosti, jelikož se nepoužívá plný název třídy, ale přímo objekt *Class*. Třídou *Class* získáme i správný typ dané instance a už se nemusíme starat o další přetypování. Tyto vlastnosti přispívají k robustnosti celé

aplikace a částečnou eliminaci chyb typu *ClassCastException*. Lookup se také používá pro správu více instancí pro jeden klíč (pro jednu třídu). Lookup umožní deklarativně přidat poskytovatele služeb, jejich vyhledání a odloženou inicializaci[3]. Komunikace mezi moduly spočívá v tom, že se předávají jednotlivé instance skrze Lookup, aniž by se jednotlivé moduly vůbec znaly. [1]

V aplikaci je také možnost využívat více než jeden objekt Lookup. Nejčastější používaný je globální Lookup, který je základní. Některé komponenty využívají také svůj vlastní Lookup – lokální (např. objekty typu TopComponent). Je dokonce možné si vytvořit vlastní objekt Lookup. [1]

Uvedeme si praktické příklady, jak se Lookup v aplikací využívá.

```
Lookup selectedContext = Utilities.actionsGlobalContext();
Openable o = selectedContext.lookup(Openable.class);

if (o != null) {
    o.open();
}
```

Na výše uvedeném příkladu je popsáno, jak získat objekt typu Openable. Podle objektu Class nám Lookup vrátí příslušnou instanci (pokud existuje, jinak vrátí null).

```
class ObjectInterestedInFooObjects implements LookupListener {
    private Lookup.Result<Foo> result;

    ObjectInterestedInFooObjects() {
        result = someLookup.lookupResult(Foo.class);
        result.addLookupListener(this);
    }

    public void resultChanged(LookupEvent evt) {
        Collection<? extends Foo> c = result.allInstances(); // do
something with the reset
    }
}
```

Tento příklad nám ukazuje použití Lookupu pro komunikaci mezi jednotlivými moduly. Stačí vytvořit třídu (můžeme využít i anonymní vnitřní třídy) a u ní implementovat rozhraní *LookupListener*. Toto rozhraní obsahuje metodu *resultChanged(LookupEvent evt)*. Ta se zavolá v případě přidání (i odebrání) objektu do Lookupu. Posluchač reaguje pouze na změny příslušného datového typu. Jednotlivé

[3] Jedná se o odložení vytváření objektu až do chvíle, kdy jej budeme potřebovat.

moduly tak mohou dynamicky reagovat na příslušné změny (jeden modul do Lookupu vloží instanci objektu a druhý modul na tuto změnu okamžitě zareaguje), aniž by měly mezi sebou závislost.

1.3.9 Okenní systém NetBeans

Okenní systém platformy je postaven na tzv. módech. Mód je třída, jež poskytuje kontejner, ve kterém se okno zobrazí. Tyto módy jsou ve výchozím nastavení rozmístěny stejně jako v NetBeans IDE. Programátor může vytvářet vlastní módy – pozice pro svá okna. Avšak pro tvorbu módů neexistuje žádný průvodce, vše je třeba udělat manuálně. [3]

Samostatná okna jsou potomky třídy TopComponent. Tato komponenta má několik stavů, které jsou pro funkci okenního systémů důležité: [1]

- **Otevřený** – komponenta je otevřená uvnitř módu
- **Uzavřený** – komponenta je zavřená nebo ještě nedošlo k jejímu otevření
- **Viditelný** – je viditelná uvnitř módu (je jediná nebo je navrchu)
- **Neviditelný** – komponenta je překrytá uvnitř módu jinou komponentou
- **Aktivní** – komponenta má fokus
- **Neaktivní** – komponenta nemá fokus

Pokud komponenty naslouchají změnám v Lookupu, není příliš vhodné, aby se tak dělo neustále (není nutné něco zobrazovat v objektu TopComponent, když jej nevidíme). Proto při otevření komponenty posluchače zaregistrujeme a při zavření je opět smažeme.

Okenní systém poskytuje i další zajímavé vlastnosti – přetahování oken do jiných modů (pozic), možnost uvolnění okna z hlavního okna aplikace, minimalizování oken do lišt, změna velikost oken apod. [1]

1.4 JAX-WS

Java API for XML Web Services (JAX-WS) je aplikační programové rozhraní pro tvorbu webových služeb na platformě Java EE. S jeho pomocí můžeme vyvíjet webové služby velice snadno a intuitivně a umožnit tak propojení serverových a klientských aplikací. Tohoto řešení se využívá například pro přístup k databázi, protože je bývá výhodnější poskytovat přístup jedno aplikaci, která následně poskytuje data všem klientům.

XML webové služby poskytují uživatelům užitečné funkce prostřednictvím standardního webového protokolu. Ve většině případů se používá protokol SOAP. Webové služby také poskytují způsob, jak dostatečně podrobně popsat svoje rozhraní, aby uživatelé mohli stavět klientské aplikace, které s nimi mohou komunikovat. Tento popis je obvykle poskytován ve formě dokumentu XML nazvaného *Web Services Description Language* (WSDL). [9]

Následující ukázka kódu zobrazuje vytvoření takové služby v jazyce Java s využitím JAX-WS.

```
@WebService(name="Calculator", serviceName="CalculatorService",
targetNamespace="http://techtip.com/jaxws/sample")
public class Calculator {

    public Calculator() {}

    @WebMethod(operationName="add", action="urn:Add")
           public int add(int i, int j) {
                  int k = i + j;
                  System.out.println(i + "+" + j + "=" + k);
                  return k;
           }
    }
```

Vývoj je jednoduchý – vytvoříme třídu, které pomocí anotace nastavím, že se jedná o webovou službu a umístíme tuto třídu do nějaké webové aplikace.

Klientská část je zobrazena v další ukázce.

```
public class JAXWSClient {
    @WebServiceRef(wsdlLocation=
        http://localhost:8080/jaxws-webservice/CalculatorService?WSDL")
    static CalculatorService service;

    public static void main(String[] args) {
        try {
            JAXWSClient client = new JAXWSClient();
            client.doTest(args);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
    public void doTest(String[] args) {
        try {

System.out.println(" Retrieving port from the service " + service);
            Calculator port = service.getCalculatorPort();
            System.out.print("Invoking add operation on the
```

```
calculatorport");

            for (int i=0;i>10;i++) {
                int ret = port.add(i, 10);
                if(ret != (i + 10)) {
                    System.out.println("Unexpected greeting " + ret);
                    return;
                }
                System.out.println("Adding:" + i + " + 10 = " + ret);
            }
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Připojení na takovou službu je také velice intuitivní. Zadáme adresu webové služby a velice snadno se potom na ni můžeme odkazovat.

1.5 SwingWorker

Swing není stavěný na vykovávání úloh náročnějších na dobu zpracování. Je prakticky koncipován jako jednovlánkový a drtivá většina objektů v něm obsažených není optimalizovaná pro používání více vláken. Programátor při použití více vláken musí brát na vědomí jistá rizika (nekonzistence paměti, deadlocks[4]). Swing využívá ke svému běhu vlákno *event dispatch thread*, ve kterém se obsluhují prakticky veškeré události a většina kódu musí být tímto vláknem vykonávána. [11]

Z toho ovšem plyne obrovské omezení, které Swing přináší. Při vykonávání úloh náročných na čas zamrzne celé GUI, dokud se úloha nevykoná. Toto řeší tzv. worker threads (pracovní vlákna). Pomocí těchto vláken můžeme vykonávat úlohy asynchronně na pozadí. Nejpoužívanější nástroj stvořený k vykonávání asynchronních úloh je SwingWorker. Je součástí Javy od verze 6.

SwingWorker je abstraktní třída, takže k jejímu využití je nutné vytvoření vlastní třídy od ní odvozené. Nejčastěji se ovšem používají anonymní vnitřní třídy. SwingWorker nabízí programátorovi několik důležitých funkcionalit, především: [19]

- Metoda *doInBackground()* je volána po spuštění workeru, zpracovává vlastní tělo úlohy
- Metoda *done()* je zavolána z hlavního vlákna swingu ihned poté, co je úloha dokončena

[4] Vlákna se nazvájem zablokují

- SwingWorker může poskytovat i mezivýsledky, konkrétně voláním metody *publish()* a *process()*

Použití třídy SwingWorker je velice jednoduché a intuitivní, což dokazuje následující ukázka:

```
private int vysledek;

private void setVysledek(int vysledek) {
    this.vysledek = vysledek;
}

new SwingWorker<Integer, Void>() {
    @Override
    public Integer doInBackground() {
        return 10;
    }

    @Override
    protected void done() {
        setVysledek(get());
    }
}.execute();
```

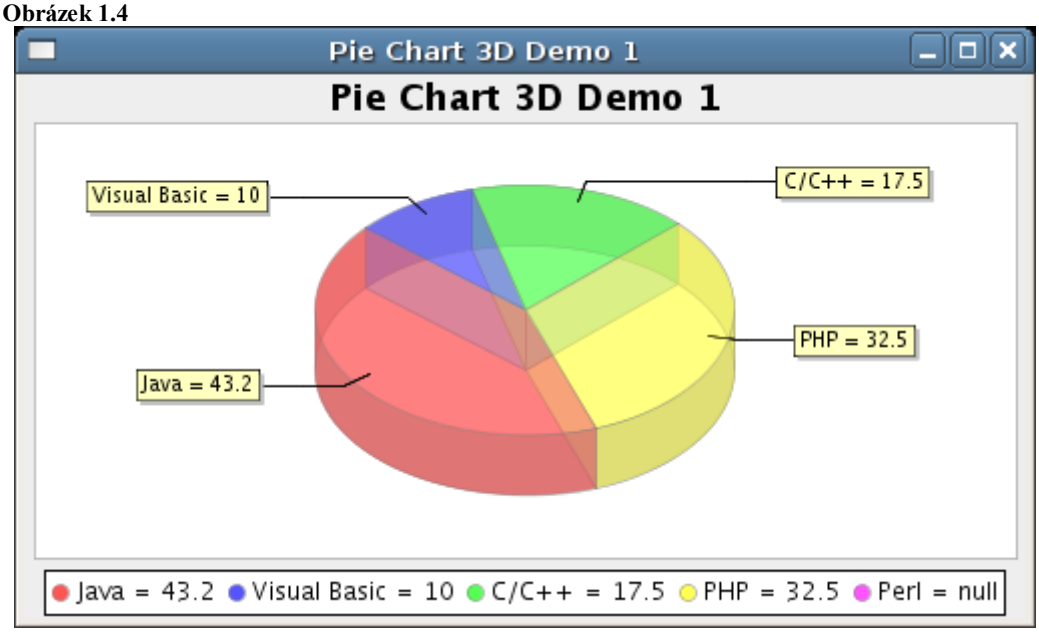
Při deklaraci je nutné uvést dva typové parametry, jak lze vidět v ukázce. První parametr určuje návratový datový typ metody *doInBackground()* a metody *get()*. Druhý typ zase určuje datový typ metody *publish()* a *process()*. Pokud nepracujeme s mezivýsledky, je zbytečné tento datový typ definovat (použijeme typ *Void*). [18]

SwingWorker se nejčastěji používá jako anonymní vnitřní třída. Je totiž zaměřený pouze na vykonání jednorázové úlohy. Pro znovuspuštění SwingWorkeru je nezbytné vytvořit zcela novou instanci – jeho použití je pouze jednorázové. Z tohoto důvodu je jeho deklarace jako anonymní vnitřní třídy nejvhodnější (existují ovšem výjimky). [18]

1.6 JFreeChart

JFreeChart je open source knihovna pro vykreslování grafů v Java aplikacích. Poskytuje konzistentní a dobře zdokumentované API podporující velkou škálu grafů. Použití tohoto API je velice intuitivní a je zaměřeno jak na serverové aplikace, tak na klientské aplikace. [10]

Obrázek 1.4



Zdroj: http://www.jfree.org/jfreechart/images/PieChart3DDemo1.png

1.7 SwingX

SwingX je kolekcí rozšířených Swingových komponent. Každá z těchto komponent poskytuje širší spektrum vlastností, než jaké jsou obsaženy ve standardním Swingu. V projektu se nachází například různé panely specializované pro zobrazování obrázků, voliče data, tabulky s možností filtrování a zvýrazňování řádků, či jiné moderní prvky, které bychom museli pro klasické swingové komponenty manuálně programovat. [17]

1.8 Apache Ant

Ant je nástroj pro řízení překladu pracující na příkazovém řádku, jehož poslání je řídit procesy spojené s kompilací. Primárně je určen pro řízení kompilování Java aplikací, avšak je možné jej použít i pro C a C++. Obecně lze pomocí Antu řídit jakýkoli proces, který může být popsán pomocí cílů (targets) a úloh (tasks). [6]

Ant je velmi flexibilní a při jeho implementaci není zapotřebí vytváření adresářových struktur či programování nějakých konvencí. [6]

Ant umožňuje programátorům psát komplexní sestavovací schémata pro své aplikace. Je celý implementován v jazyce Java, takže není závislý na konkrétním operačním systému. To jej předurčuje k rozsáhlému využití. Prakticky každá operace je implementována za pomoci standardních javovských knihoven, tedy bez používání nativních aplikací operačního systému. Ant pro svůj běh využívá skripty (sestavovací

soubory), které jsou psány v XML formátu. To dovoluje kontrolovat samotné sestavovací schéma. [8]

```
<project name="Priklad_2" default="compile">
    <target name="compile">
        <mkdir dir="bin"/>
        <chmod file="bin" perm="a+xr"/>
        <javac srcdir="src" destdir="bin"/>
    </target>

    <target name="clean">
        <delete dir="bin"/>
        </target>
    </project>
```

Pokud Ant provede výše uvedený skript, tak se provede implicitní cíl *compile*. Tento cíl má za úkol vytvořit složku *bin*, nastavit ji uživatelská práva a říci kompilátoru, kde hledat zdrojové kódy a kde ukládat zkompilované kódy. Pokud Ant spustíme s argumentem *clean*, provede se cíl clean, který maže adresář *bin*. Tato vlastnost nám umožňuje definovat ve skriptu mnoho cílů a přepínačem si cíle vybírat. Celý proces můžeme sestavit do několika fází a ty mohou být na sobě závislé (například kompilace knihoven a následná kompilace aplikace, případně její umístění na aplikační server). [8]

1.9 Apache Maven

Maven je kompilační nástroj, jehož původním účelem bylo zjednodušení kompilace rozsáhlých Java aplikací. Na rozdíl od nástroje Ant funguje na zcela opačné filozofii. Maven funguje na principu deklarativního programování – popíšeme „co chceme udělat“ a Maven se o průběh postará sám (obdoba SQL[5] a DBMS[6]). Maven vyhledává jednotlivé objekty v tzv. repositáři, do kterého také ukládá veškeré zkompilované projekty. Tento adresář má klasickou javovskou balíčkovou strukturu. Pokud má projekt nastavenou nějakou závislost a příslušný projekt se v repositáři nenachází, skončí kompilace chybou. [13]

Maven je modulárně vyvíjený framework. Maven přímo není zodpovědný za proces kompilace, případně dalších úloh (generování dokumentace, umístění projektu na aplikační server apod.). Veškerou práci vykovávají jednotlivé moduly, které Maven spravuje. Existují různé moduly, například pro generaci dokumentace v PDF formátu, případně kompilace a zabalení projektu do JAR, RAR nebo WAR archívů. Modul

[5] Standard Query Language
[6] Database Management System

umožňující kompilaci aplikace vyvíjené na platformě NetBeans se nazývá *nbm-maven-plugin*. [12]

Repozitář se dělí na dvě úrovně – **lokální** a **vzdálený**. **Lokální repozitář** se ve výchozím nastavení nachází v domovském adresáři uživatele v podsložce *.m2/repository*. Tento repozitář obsahuje veškeré lokálně zkompilované projekty. Do tohoto repozitáře se také nahrávají projekty ze vzdálených repozitářů, kvůli rychlejšímu přístupu. **Vzdálený repozitář** je většinou umístěn na internetu a obsahuje obrovské množství projektů. URL adresy těchto repozitářů si musíme definovat v nastavení Mavenu. Drtivá většina vývojářů doporučuje Maven k získání aktuálních verzí svých projektů. [2]

Maven využívá tzv. *Project Object Model*, pomocí kterého se projekt popíše. Princip je velice jednoduchý. Vše je založeno na XML dokumentu, který má název *pom.xml* a nachází se v kořenovém adresáři projektu. Zkompilovaný projekt se uloží do lokálního repozitáře, ve kterém se, v případě potřeby (je na něj nastavena závislost u jiného projektu), vyhledá v příslušné verzi. Pokud není nalezen v repositáři lokálním, začne Maven vyhledávat v repozitářích definovaných ve svém nastavení. [13]

Význam jednotlivých elementů v *pom.xml*:

- **modelVersion** – verze POMu
- **groupId** – název balíčku, slouží k uložení do repozitáře a následně jeho vyhledávání
- **artifactId** – název artefaktu (název pod kterým se vyhledá v repositáři)
- **version** – verze projektu
- **dependencies** – zde se uvedou jednotlivé závislosti; artefakty se nejdříve vyhledají v lokálním repositáři a až poté v repozitářích veřejných; vyhledávají se závislosti v odkazovaných projektech

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.mycompany.app</groupId>
    <artifactId>my-app</artifactId>
    <version>1.0</version>

    <dependencies>
        <dependency>
            <groupId>com.mycompany.app2</groupId>
            <artifactId>my-app2</artifactId>
            <version>1.0.3</version>
        </dependency>
```

```
    </dependencies>
</project>
```

Maven poskytuje jednoduchou a velice intuitivní správu vývoje velkých projektů. Vývojové prostředí NetBeans poskytuje výbornou podporu pro Maven (NetBeans 7.0 již podporuje Maven3). [14]

1.10 Apache Tomcat

Tomcat je open-source webový kontejner vyvíjený společností Apache Software Foundation. Vzhledem k pouze částečné implementaci Java EE je velice rychlý a transparentní, díky čemuž si získal obrovské množství uživatelů po celém světě.

Webový kontejner je rozhraním mezi webovými komponentami a webovým serverem. Webovou komponentou rozumíme servlet, JSP nebo JSF stránku. Tento kontejner řídí životní cyklus těchto komponent a předává jim určité požadavky. [16]

1.11 Spring Framework

Spring Framework poskytuje komplexní infrastrukturu pro vývoj Java aplikací. Jeho hlavní myšlenkou je zjednodušit práci programátorovi tím, že se může zaměřit pouze na svou aplikaci, místo vymýšlení celé logiky. Spring umožňuje vyvíjet *enterprise* aplikace pouze s použitím tzv. POJOs[7]. Spring obsahuje kolem 20 modulů, které jsou rozčleněny do větších skupin. [5]

Spring se velice často používá k deklarativnímu spravování Java Beans s možností přístupu do databáze skrz abstraktní vrstvu (ať už využíváme JDBC či ORM).

[7] Plain Old Java Objects

2 Analýza stávajícího stavu

2.1 Popis společnosti

ON Semiconductor je mezinárodní společností se sídlem ve městě Phoenix v americkém státě Arizona. Je jedním z předních světových výrobců integrovaných obvodů a diskrétních polovodičových součástek, které jsou používány v nejrůznějších elektronických zařízeních.

České společnosti skupiny ON Semiconductor navazují na již šedesátiletou tradici v oboru elektroniky a na své historické předchůdce Tesla Rožnov (1949-1991), Tesla Sezam a Terosil. Moderní integrované obvody se dnes navrhují a vyrábí ve společnostech SCG Czech Design Center, ON Design Czech a ON Semiconductor Czech Republic. Mezi hlavní aktivity společnosti patří také návrh integrovaných obvodů, výzkum a vývoj, výroba křemíku, výroba čipů a také sdílené IT služby.

IT centrum sdílených služeb bylo založeno v roce 1999 a je organizačně zařazeno do společnosti SCG Czech Design Center. Skupina specialistů tohoto IT centra se zabývá vývojem, nasazováním a podporou informačních systémů pro potřeby výrobních závodů nadnárodní skupiny ON Semiconductor v oblasti řízení výroby, zpracování dat, automatizace, implementace a podpory ERP, BI, podpory databází a IT infrastruktury atp.

Aktivity IT centra sdílených služeb jsou řízeny potřebami našeho hlavního zákazníka - mateřské společnosti, která určuje strategii nasazení a podpory IT systémů ve výrobních závodech ON Semiconductor.

Tým IT centra sdílených služeb má v současné době asi 50 zaměstnanců a v příštích několika letech je plánována jeho další expanze. V regionu se tak řadí mezi největší IT zaměstnavatele.

2.2 Úlohy, které aplikace plní

Aplikace slouží zaměstnancům při výrobě polovodičových desek. Jedná se o Rich-Client aplikaci, která se využívá při výrobě polovodičů. Veškerá data jsou uložena v repositáři, který je druhou části této aplikace a jedná se o část serverovou (ve skutečnosti jsou to dvě samostatné, avšak úzce spjaté aplikace). Aplikace také umožňuje upload nových dat na server a ukládání dat do lokálního počítače.

Tento systém využívají především pracovníci výroby. Ti z něj získávají informace o kvalitě vyrobených čipů (binů). Tato aplikace je tedy pro zlepšování efektivity a kvality výrobků velice důležitá. Neméně důležitá je i zpětná vazba od zaměstnanců, kteří při intuitivní práci s moderním systémem dosahují daleko lepších výsledků, proto je kladen velký důraz na neustálý vývoj a modernizaci.

2.3 Financování projektu

Společnost ON Semiconductor je mezinárodní korporací vlastnící vývojová centra v Evropě i Asii. Z hlediska finančních zdrojů je projekt zabezpečen díky inovativní politice společnosti. Investované částky se velice rychle vrátí, především díky zvyšování efektivnosti práce a spokojenosti zaměstnanců. Společnost si také velice dobře uvědomuje, že bez pořádné IT infrastruktury je konkurenceschopnost na moderním trhu prakticky nulová.

U lidských zdrojů je třeba brát v potaz, že aktuální situace ve společnosti vypovídá o velikém zájmu o IT služby. Evropské vývojové centrum nabírá zkušenou pracovní sílu a spolupracuje s vysokými školami ve věci odborných stáží a stipendií pro studenty.

Společnost využívá pro interní komunikaci a IT infrastrukturu virtuální privátní síť, která se rozprostírá po všech pobočkách na celém světě. Zaměstnanci tak mohou využívat IS, ať už se nacházejí v Americe, Evropě či Asii. Distribuce se stává velice jednoduchou a náklady se o to snižují.

2.4 Definice informačních výstupů a funkcí

Informačním výstupem z hlediska obsahu jsou jednotlivé mapy křemíkových desek (waferů). Každá mapa má své specifické údaje, podle kterých může být v systému dohledána. Nové mapy vznikají ve výrobním systému a upravené mapy vkládají pracovníci.

Z hlediska formy se jedná o Rich-Client aplikaci, která se připojuje na server do repozitáře, odkud se hledají příslušné mapy, které uživatelé vyžadují.

Zdrojem dat jsou mapy, které přicházejí z výrobního systému při výrobě polovodičů. Tyto mapy mohou operátoři výroby nadále upravovat podle aktuálních požadavků a nahrávat je do repozitáře.

Společnost ON Semiconductor poskytuje celosvětově tuto aplikaci pomocí intranetu a technologie Java Web Start. Na různých úrovních je odfiltrováno používání systému neprivilegovanými osobami. Dále existuje systém práv pro uživatele, který jednotlivým pracovníkům definuje právo na úpravy a mazání jednotlivých map.

Aplikace je primárně určena pro zaměstnance výroby. Jedná se zcela o interní aplikaci, ke které nemají přístup lidé mimo společnost. Přístup k systému je velice důležité dobře zabezpečit, aby se veřejnost či konkurence nedostala k know-how společnosti.

2.5 Vyhodnocení, zda již existuje využitelný projekt

Aplikace je vyvíjená přímo pro potřeby společnosti ON Semiconductor. Vzhledem k tomu, že ji vyvíjí interní vývojový tým, nebylo evidentně nalezeno vhodné externí řešení. Jedná se o velice specifický typ aplikace, která nemá široké uplatnění, proto pochybuji, že by tento typ aplikace byl vyvíjen nějakou jinou společností a následně umístěn na softwarový trh.

Vývojový tým v minulosti vytvořil podobnou aplikaci, která ale nabízí podstatně jiné funkce. Je především daleko rozsáhlejší, náročnější na systém a na uživatelské ovládání.

2.6 Informační vazby, jež je nutno dodržovat

V projektu existují dva typy informačních vazeb.

Zaprvé se jedná o výrobní systém. Tento systém při výrobě vytváří mapy jednotlivých waferů a tyto mapy se dále nahrávají do repozitáře. Toto je naprosto stěžejní činnost, na které celý systém stojí.

Zadruhé se jedná o operátory výroby, kteří pracují s aplikací, pomocí které si zobrazují obsah repozitáře. Tito pracovníci zkoumají množství chyb na vyrobených polovodičích a zajišťují optimální výrobní metody vedoucí k zajištění nejvyšší možné kvality vyrobených součástek.

2.7 Datová základna projektu

2.7.1 Souhrn

Systém je tvořen ze dvou aplikací – klientské a serverové. Serverová aplikace funguje jako repozitář jednotlivých map a obstarává uchovávání informací. Pro komunikaci s klientem jsou využity webové služby. Klient funguje jako prohlížeč dat

v repositáři, ale může zobrazovat i lokálně uložené mapy. V možnostech klienta je i upload lokálních map do repozitáře. Repozitář využívá databázi Oracle k uchovávání informací o jednotlivých mapách.

2.7.2 Infrastruktura repozitáře

Serverová část strukturovaně ukládá soubory map do souborového systému a informace o jednotlivých souborech (mapách) do databáze. Soubory jsou nahrány do repozitáře skrze webové služby, které poskytují i metody pro získávání map z repozitáře. Důležitá je i funkce vyhledávání, která klientovi umožňuje velice jednoduše prohledávat repozitář.

Pokud při uploadu dojde ke kolizi (nahraje se mapa, která v databázi již existuje), tak přijde na řadu verzovací systém. Ten zaručí označení nové mapy vyšší verzí a následnému uložení již nic nestojí v cestě.

2.7.3 Souborový systém repozitáře

Každá mapa je ukládána v samostatném souboru, který je zkomprimován do ZIP archivu. Každý soubor je přesně pojmenován podle svého Lot ID (ID celé série) pro lepší orientaci v souborovém systému.

V repositáři existují tři úrovně adresářů. První úroveň je definována jako prvních pět znaků Lot ID. Druhá úroveň je definována jako prvních sedm znaků a třetí úroveň již obsahuje celé Lot ID. Tato struktura velice zpřehledňuje uchovávání velkých objemů dat.

Například všechny mapy s Lot ID "**ABCD123456**" jsou uloženy v adresáři "**/ABCD1/ABCD123/ABCD123456/**"

V tomto adresáři jsou uchovány veškeré verze této mapy pod patřičným unikátním názvem souboru.

2.7.4 Databázová část

Systém využívá databázi pro hledání v repositáři a uchovávání nezbytných informací o jednotlivých mapách.

V databázi existují tři hlavní tabulky s informacemi o jednotlivých souborech a mnoho vedlejších s doplňujícími informacemi o jednotlivých mapách.

Tabulka MS_LOT uchovává informace o lotu, technologiích výroby atd. Tabulka MS_WAFER drží informace o jednotlivých waferech a tabulka

MS_MAP_FILE uchovává informace o jednotlivých souborech map (viz příloha č. 1, obrázek 2.1).

2.7.5 Zabezpečení dat

Data jsou na serverech velice dobře zabezpečena. Souborové servery využívají disková pole a databázové servery pravidelně ukládají zálohy. Veškerá data jsou následně archivována na magnetické pásky, které se uchovávají na utajených místech odolných proti živelným pohromám.

2.8 Definování prostředí pro provozování projektu

Serverová aplikace běží na aplikačním serveru a používá databázi a souborový systém serveru. Klientská aplikace běží na uživatelských PC. Obě části jsou napsány v jazyce Java. Serverová část využívá Apache Tomcat pro běh aplikace a DB Oracle. Je třeba mít minimálně dva fyzické servery – jeden pro aplikaci, jeden pro DB. V případě vysokého vytížení je vhodné replikovat DB na další servery a využívat i externí fileserver pro ukládání map. Na aplikačním serveru je třeba mít nainstalovanou Javu ve verzi 6.0.

U serverové části je také nezbytné počítat s velikostí repozitáře. Ačkoliv jsou jednotlivé soubory komprimovány, tak při velkém množství můžou zabírat mnoho GB. Je proto důležité využívat robustní a veliké diskové pole.

Vzhledem k nezávislosti Javy na operačním systému je možné mít na klientských počítačích prakticky libovolný moderní OS (MS Windows XP, Vista, Linux apod.). Je ovšem důležité zmínit, že aplikace jako taková je velice robustní a tomu odpovídá náročnost na operační paměť počítače v závislosti na velikosti mapy. Ta se pohybuje v rozmezí 30MB – 120MB. Pro spuštění klienta je používána technologie Java Web Start, která zaručuje používání nejnovější verze aplikace na všech počítačích. Odpadají také instalace na klientské počítače. Uživatel spustí aplikaci z intranetu, do kterého se přihlásí pod svým ID.

Důležité je také zmínit vytížení CPU při zobrazování jednotlivých map. Aplikace může na jistý moment vytížit procesor na 100% při zobrazování enormně velké mapy.

Aplikace byla testována na počítačích s různou hardwarovou konfigurací. Od starých jednoprocesorových počítačů s 256MB RAM až po moderní víceprocesorové

pracovní stanice. Testy byly především zaměřeny na paralelní zpracování dat v aplikaci (velký nárůst výkonu u víceprocesorových stanic).

2.9 Struktura Rich-Client aplikace

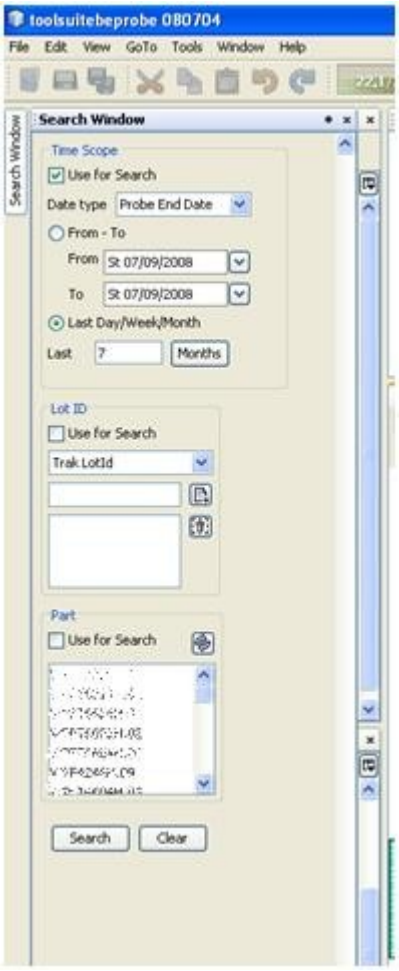
2.9.1 Obecné informace

Celá aplikace je postavena na platformě NetBeans. Na úvod je nutné podotknout, že swingové komponenty nejsou programovány přímo v jednotlivých modulech, ale v samostatných projektech. Důvodem bylo případné využití těchto komponent (panelů, tabulek apod.) v jiných aplikacích. Aplikace byla kompilována pomocí nástroje Apache Ant a pro každou externí knihovnu (jednotlivé komponenty GUI, JAX-WS, Spring, SwingWorker, Log4J atd.) byl vytvořen vlastní modul, aby mohla být využívána na úrovni platformy. To vyústilo ve velké množství modulů (téměř 40). Komunikaci mezi moduly zprostředkovává Lookup (viz kapitola 1.3.8). Do Lookupu se vkládají instance různých Java Beans, které se nacházejí v modulu *databeans*. Tento modul obsahuje veškeré Java Beans, které aplikace využívá, včetně několika dalších tříd případně rozhraní (například poskytovatele ukládání a uploadu map). Na tomto modulu jsou závislé všechny další moduly v aplikaci. Nejdůležitějším modulům a komponentám jsou níže věnovány samostatné podkapitoly (z drtivé většiny se jedná o okna aplikace – objekty typu TopComponent). Screenshot celé aplikace se nachází v příloze č. 3, obrázek 2.2.

2.9.2 Search TopComponent

Toto okno je ve výchozím nastavení minimalizováno do panelu na levé straně aplikace. Slouží uživatelům k vyhledávání potřebných dat. Výsledky hledáni jsou omezeny třemi kritérii (každé lze aktivovat zatržením příslušného checkboxu, mohou být kombinovány):

Obrázek 2.3: Search Window



Zdroj: [Interní zdroj]

- **Time Scope (časové rozlišení)**

  Zde může uživatel stanovit časový interval, přičemž má dvě možnosti. Zaprvé si může vybrat období od-do (pomocí uživatelsky přívětivých voličů data, tzv. datepickers), zadruhé může zvolit počet posledních dnů/týdnů/měsíců pro odfiltrování starších dat.

- **Lot Id (číslo lotu)**

  Uživatelé mohou vyhledávat i podle jedinečného čísla celé sady (lot Id).

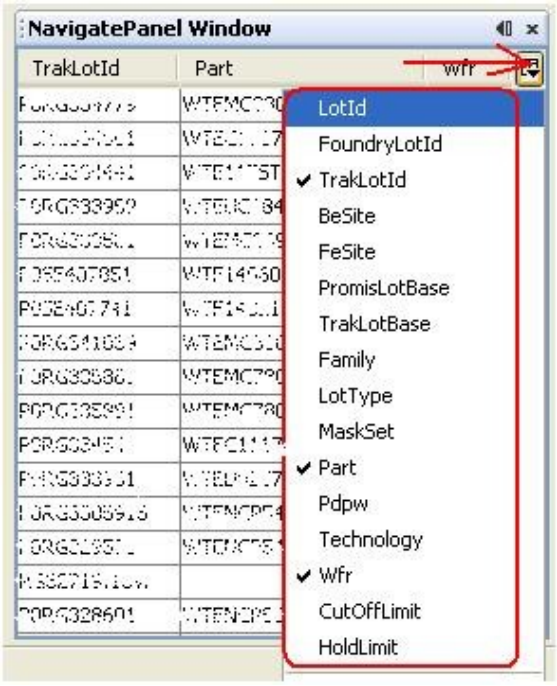
- **Part (typ výrobku)**

  V případě potřeby vyhledání všech lotů jednoho typu výrobku, je možné využít panel Part k vyhledání daného typu.

2.9.3 Navigate TopComponent

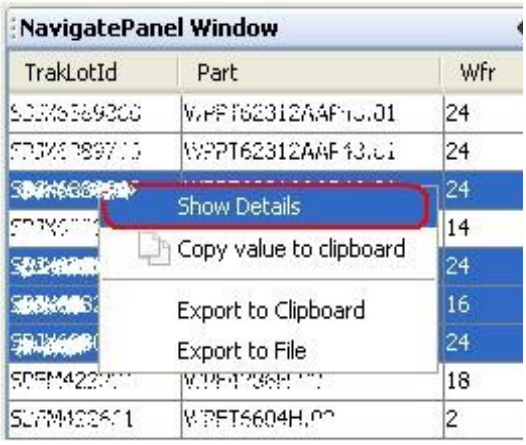
Tento panel zobrazuje v tabulce výsledky hledání. Ve výchozím nastavení se zobrazují tři údaje (Lot Id, Part a Wafer). Uživatel si ovšem může zobrazit další sloupce kliknutím na tlačítko v pravém horním rohu.

Obrázek 2.4: NavigatePanel Window



Zdroj: [Interní zdroj]

Obrázek 2.5: NavigatePanel Window 2



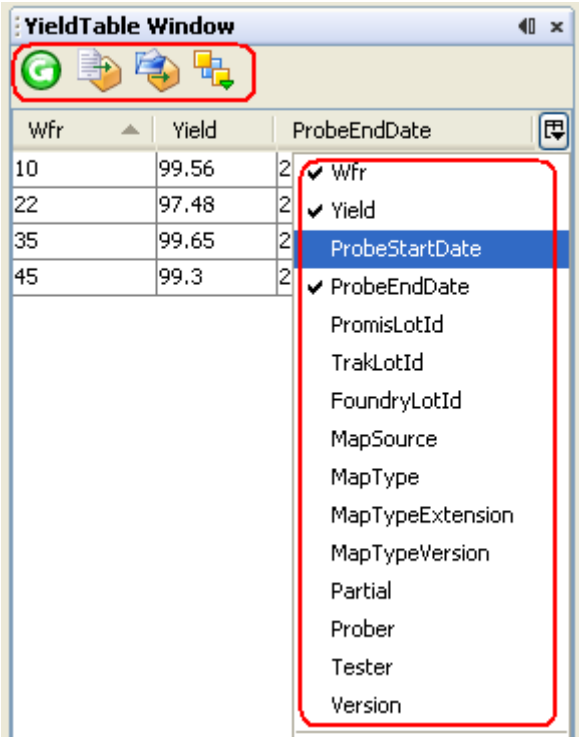
Zdroj: [Interní zdroj]

Uživatel si může vybrat více než jeden lot. Při výběru jednoho lotu lze na položku pouze kliknout, jinak se musí označit více položek a z kontextové nabídky vybrat příkaz „Show Details“. Při výběru lotu se do Lookupu přidá příslušný Java Bean, na který čekají dvě komponenty – YieldTable a OnChart. Obě se při výběru aktualizují.

2.9.4 Yield Table TopComponent

Tato tabulka zobrazuje informace o změřených waferech příslušného (vybraného) lotu. Opět je ve výchozím nastavení viditelná pouze část údajů, jejíž počet si může uživatel sám měnit. Nad tabulkou se nachází panel s tlačítky (obnovení, zobrazení a filtrování). V závislosti na datech (výsledcích měření) se jednotlivé řádky barevně vybarvují (bílá – v pořádku, žlutá – nižší kvalita, červená – nízká kvalita). Při výběru se do Lookupu přidají objekt(y), na které reaguje komponenta WaferMapDetail.

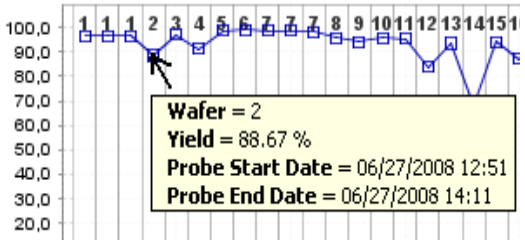
Obrázek 2.6: YieldTable Window



Zdroj: [Interní zdroj]

2.9.5 OnChart TopComponent

Obrázek 2.7: OnChart Window

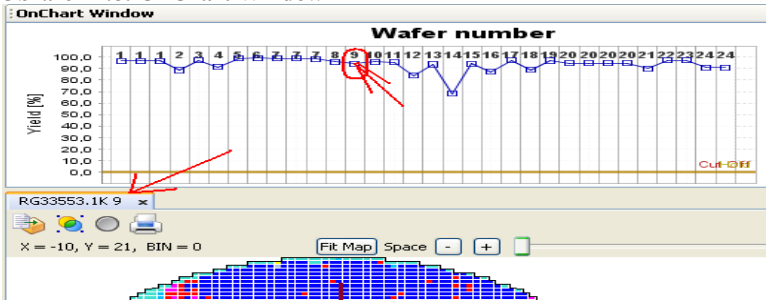


Zdroj: [Interní zdroj]

V této komponentě se nachází graf, který plní podobnou funkci jako komponenta YieldTable. Zobrazuje jednotlivé wafery v grafu. Jedná se o liniový graf, který zobrazuje informace při najetí kurzoru na příslušné body.

Při kliknutí zobrazí v komponentě WaferMapDetail příslušný wafer (přidá do Lookupu stejný objekt, jako YieldTable).

Obrázek 2.8: OnChart Window 2



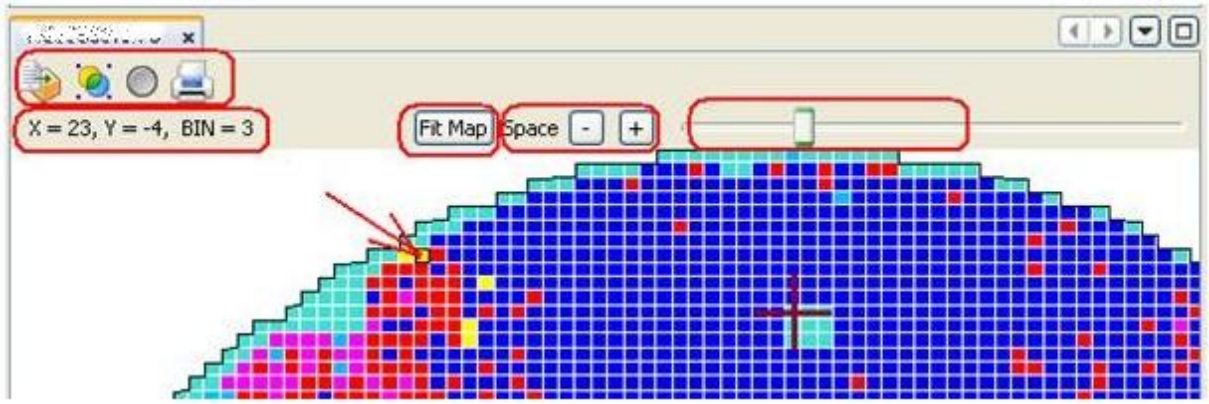
Zdroj: [Interní zdroj]

2.9.6 WaferMapDetail TopComponent

Tato komponenta je srdcem aplikace. Ukazuje totiž uživateli jednotlivé čipy (biny) na waferu (mapa waferu). Celá mapa není pouze statická, ale uživatel si může jednotlivé biny vybrat klikem myši a komponenta mu zobrazí jeho souřadnice a hodnotu. Horní panel obsahuje posuvný jezdec (JSlider) pro přibližování a oddalování mapy, tlačítka plus a mínus pro zvětšování a zmenšování mezer mezi čipy a nakonec tlačítko Fit Map. To vycentruje a roztáhne mapu do celého okna.

Vlevo v nástrojové liště jsou čtyři tlačítka. První tři zleva přepínají různé pohledy – standardní, s vysoce kontrastníma barvami a prázdnou mapu.

Obrázek 2.9: WaferMapDetail Window



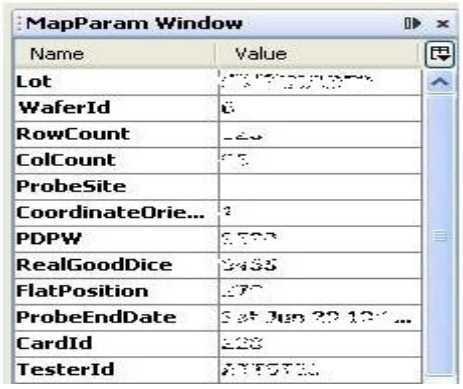
Zdroj: [Interní zdroj]

Tato komponenta využívá vlastností editoru v NetBeans. Všechna okna se sdružují do záložek, takže nic nebrání v otevření více oken a přepínání mezi nimi. Uživatel může zavřít i všechny najednou nebo všechny kromě aktuálně aktivní záložky.

2.9.7 MapParam TopComponent

Tato tabulka zobrazuje jednotlivé parametry mapy. Nejdůležitější položky jsou umístěny nahoře a jsou tučně zvýrazněny.

Obrázek 2.10: MapParam Window

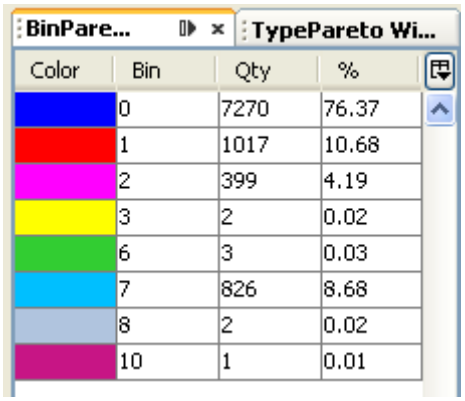


Zdroj: [Interní zdroj]

2.9.8 BinPareto TopComponent

BinPareto zobrazuje informace o konkrétních čipech na desce. Najdeme zde údaje o počtu čipů na desce, jejich procentuální výskyt a jiné ukazatele. Uživatelé mohou měnit barvy jednotlivých binů pomocí speciálního dialogu pro výběr barvy (je dostupná z kontextové nabídky), případně mohou jednotlivé biny zobrazovat či schovávat. Vše se následně projevuje v komponentě WaferMapDetail.

Obrázek 2.11: BinPareto Window

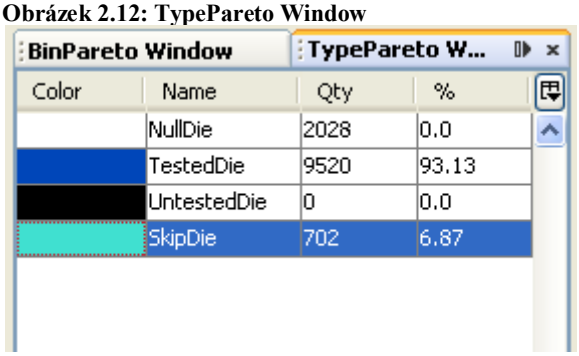


Zdroj: [Interní zdroj]

2.9.9 TypePareto TopComponent

Tato komponenta je velice podobná komponentě BinPareto. Nezobrazuje ovšem údaje o jednotlivých čipech, ale o typech čipů.

Obrázek 2.12: TypePareto Window



Zdroj: [Interní zdroj]

2.9.10 Toolbar

V nástrojové liště jsou umístěny ikony pro upload a ukládání map. Jejich umístění a obslužné třídy jsou popsány v layer.xml (viz ukázka níže).

```
<filesystem>
    <folder name="Actions">
        <folder name="System">
            <file name="com-onsemi-cim-apps-toolsuitebe-toolbar-
SaveWaferMapAction.instance"/>
            <file name="com-onsemi-cim-apps-toolsuitebe-toolbar-
UploadWaferMapAction.instance"/>
        </folder>
    </folder>
    <folder name="Menu">
```

```
        <folder name="File">
            <file name="com-onsemi-cim-apps-toolsuitebe-toolbar-
SaveWaferMapAction.shadow">
                <attr name="originalFile"
stringvalue="Actions/System/com-onsemi-cim-apps-toolsuitebe-
toolbar-SaveWaferMapAction.instance"/>
                <attr name="position" intvalue="1350"/>
            </file>
       ...
</filesystem>
```

2.9.11 Modul banner

Tento modul obsahuje veškeré informace o polohách všech TopComponent. V souboru layer.xml jsou definovány cesty k souborům *.wsmode*, které obsahují údaje o pozicích těchto modů. V souboru layer.xml příslušné TopComponenty se definuje, v jakém modu (tudíž na jaké pozici) tuto komponentu chceme otevřít.

```
<filesystem>
    <folder name="Windows2">
        <folder name="Modes">
            <file name="explorer.wsmode"
url="modes/explorer.wsmode"/>
            <file name="navigator.wsmode"
url="modes/navigator.wsmode"/>
            <file name="properties.wsmode"
url="modes/properties.wsmode"/>
            <file name="pareto.wsmode"
url="modes/pareto.wsmode"/>
            <file name="banner.wsmode"
url="modes/banner.wsmode"/>
        </folder>
    </folder>
</filesystem>
```

2.10 Struktura serverové části

2.10.1 Obecně

Serverová aplikace je klasická Java EE 1.5 aplikace, která ke svému běhu využívá webový kontejner Apache Tomcat ve verzi 6. Jejím úkolem je dělat prostředníka mezi databází a Rich-Client aplikací. Poskytuje klientovi veškerá potřebná data a ukládá potřebná data do databáze a repozitáře.

2.10.2 Spring Framework

Pro správu Java Beans a se využívá Spring Framework. Umožňuje konfigurovat Java Beans deklarativně, což je velice výhodné.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-
2.5.xsd">

    .....

    <bean id="ToolsuiteFileSystem"
class="com.onsemi.cim.apps.toolsuite.be.data.FileSystemImpl" >
        <property name="storagePath"  value="${tsbe.rootpath}"
/>
        <property name="rejectPath"   value="${tsbe.rejectpath}"
/>
    </bean>

    <bean id="Cache"
class="com.onsemi.cim.apps.toolsuite.be.data.WaferMapJrCache" >
        <property name="cachePath" value="${tsbe.cachepath}" />
        <property name="repositoryRoot" value="${tsbe.rootpath}"
/>
    </bean>

    .....

</beans>
```

Podle toho, kde bude aplikace nasazena, se při kompilaci podsune příslušný konfigurační soubor, který obsahuje informace o databázi a cestě k ukládaným mapám.

```
#toolsuite BE data source connection
tsbe.driver=oracle.jdbc.driver.OracleDriver
tsbe.url=jdbc:oracle:thin:@***.ONSEMI.COM:1521:SYN6
tsbe.user=***
tsbe.password=***

# apps paths config
tsbe.rootpath=D:/MyTemp/map-file-repository/maps
tsbe.cachepath=D:/MyTemp/map-file-repository/cache
tsbe.rejectpath=D:/MyTemp/map-file-repository/rejects
```

Jednotlivé Spring Beans (Java Beans spravované pomocí Spring Frameworku) lze snadno získat v datovém typu dle našeho přání. Stačí využít generiky a vrátit bean jako datový typ objektu Class v parametru metody.

```
public static<T> T getDAO(DAOs d, Class<T> c){
        ApplicationContextHolder ctx =
ApplicationContextHolder.getInstance();
        switch(d){
            case Lot:
                return (T)ctx.getBean(c, "LotDAO"); // bylo
codebookdao.class
            case Wafer:
                return (T)ctx.getBean(c, "WaferDAO");
            case MapFile:
                return (T)ctx.getBean(c, "MapFileDAO");
            case ToolsuiteBE:
                return (T)ctx.getBean(c, "ToolsuiteBE");
            case ProcessErrorLog:
                return (T)ctx.getBean(c, "ProcessErrorLog");
            case ErrorLog:
                return (T)ctx.getBean(c, "ErrorLogDAO");
            case Cache:
                return (T)ctx.getBean(c, "Cache");
            default:
                throw new RuntimeException("Unknown type of
DAO");
        }
}
```

2.10.3 Webové služby

Webové služby zajišťují veškerou komunikaci mezi klientem a serverem. Ukázka takové webové služby:

```
@WebMethod(operationName = "getMapFiles")
    public List<MapFile> getMapFiles(@WebParam(name = "filter")
QueryToken filter) {

        log.debug("GetMapFiles web service invoked");
        MapFileDAO dao = DAOUtils.getDAO(DAOs.MapFile,
MapFileDAO.class);
        return dao.getAll(filter);
    }
```

3 Implementace nové funkcionality

3.1 Přechod z Apache Ant na Apache Maven2

V první řadě se přecházelo z Apache Ant na Apache Maven2. Přechod byl velmi výhodný už jen proto, že má společnost interní repozitář, do kterého ukládají své projekty různé interní vývojové týmy. Závislosti jednotlivých modulů se přechodu nastavují přímo na úrovni Mavenu a to konkrétně v *pom.xml* (čili stejně jako při využívání knihoven). K integrování swingových komponent využívaných v aplikaci byl vytvořen nový modul „gui-api“, který má nastavené závislosti na všechny potřebné JAR soubory, které tyto komponenty obsahují (viz kapitola 2.8.1). Na tomto modulu jsou pak závislé ostatní moduly, které pro svůj běh tyto knihovny využívají.

Přechod na Maven ovšem nebyl zcela bezbolestný. Vyskytly se drobné problémy například při podsunutí upraveného konfiguračního souboru platformy (v tomto souboru se nastavuje domovský adresář aplikace, velikost haldy a jiné údaje, které definují pracovní prostředí aplikace). Problém nakonec šel vyřešit velice elegantně, protože Maven (resp. nbm-maven-plugin) s tímto přímo počítá. V *pom.xml* běhového kontejneru lze přímo definovat cestu k našemu konfiguračnímu souboru *.conf*, a to v tagu **<etcConfFile></etcConfFile>**. Plugin při kompilaci následně platformě tento soubor podsune místo výchozího. Soubor následně umístit do příslušného balíčku.

Přechod na Apache Maven2 měl sice na starost jiný člen vývojového týmu, avšak je to poměrně radikální změna a by byla škoda ji nezmínit.

3.2 Modul WaferGallery

Na základě uživatelských požadavků bylo zapotřebí přidat do aplikace funkcionalitu, která by umožňovala hromadné prohlížení jednotlivých desek v lotech (sadách). Jednoduše řečeno bylo zapotřebí vytvořit galerii, která by mimo údaje poskytovaných moduly YieldTable a OnChart (viz kapitola 2.8.5 a 2.8.4) poskytovala i náhled jednotlivých desek. Uživatelé dále požadovali, aby mohli v galerii prohlížet jednotlivé desky, které si sami vyberou, případně celé sady (také dle vlastního výběru).

Bylo nutné vymyslet jakým způsobem tyto náhledy poskytovat klientovi. Řešení na úrovni platformy bylo vcelku jednoduché – vytvořit nový modul, který by tato data zpracoval.

3.2.1 Klientská část

Do aplikace byl přidán nový modul, který obsahuje jedinou třídu, její rodičovská třída je třída TopComponent a implementuje rozhraní LookupListener. V souboru *layer.xml* bylo nastaveno, že se má komponenta zobrazovat na pravé straně aplikace a má být minimalizována do pravé lišty. Toto není třeba definovat manuálně, průvodce to automaticky nastaví za nás.

Pro potřeby modulu byl vytvořen nový Java Bean (WaferGalleryBean), který obsahuje dva seznamy:

- **Seznam sad (lotList)** – tento seznam obsahuje údaje o celých sadách, které si uživatel přeje zobrazit
- **Seznam desek (mapFileList)** – v tomto seznamu jsou uloženy údaje pouze pro jednotlivé desky, které si uživatel přeje zobrazit

Pokud si uživatel přeje zobrazit sady či desky v galerii (učiní tak nějakou uživatelskou akcí, např. kliknutím na tlačítko), přidá se do Lookupu WaferGalleryBean s konkrétními daty. Na tuto událost okamžitě zareaguje modul WaferGallery a dotáže se na server pro příslušné náhledy, podle dat která obdržel z Lookupu. Následuje načtení příslušných náhledů ze serveru.

Vzhledem k tomu, že komunikace mezi klientem a serverem je časově náročná (závisí na rychlosti sítě, vytíženosti serveru a jiných faktorech), je důležité, aby se tak dělo na pozadí v samostatném vláknu. Ideální pro tento případ je SwingWorker (viz kapitola 1.5), který obstará celou komunikaci a načítání náhledů. Uživatel poté nepocítí zpomalení či zatuhnutí samotné aplikace v době načítání. SwingWorker je ale konstruovaný pouze pro jednorázové použití, což jej předurčuje pro používání jakožto anonymní vnitřní třídy. Problém nastává ve chvíli, kdy se načítají do galerie data, a uživatel se rozhodne načíst jiná data. V tomto případě musíme mít uloženou instanci SwingWorkeru v nějaké proměnné, abychom mohli v případě potřeby jeho práci stornovat. Při zadání jakékoli nové práce je důležité této proměnné přiřadit zcela novou instanci. Ukázka této situace:

```
if (worker != null) {
    worker.cancel(true);
    worker = null;
}
```

```
worker = new IndicableSwingWorker(this, null) {
    @Override  protected Object doWork() {
        System.out.println("refreshing wafer galery ...");
        setGalleryContent(mapFileListBean); //vlastní načítání
dat do galerie
        return null;
    }
};

worker.execute();
```

3.2.2 Serverová část

Existuje několik variant, jak poskytovat klientovi příslušné náhledy desek. Jako nejvhodnější se nakonec ukázalo vygenerování obrázku (ve formátu PNG) a následné uložení do souborového systému serveru, aby v případě další potřeby, nebyl obrázek opět generován. Bylo možné využít webové služby, pro poslání obrázku klientovi, ovšem je nutné počítat se serializací a následnou deserializací obrázku, což má negativní dopad na efektivitu zpracování. Daleko vhodnější bylo vytvořit servlet, který by na základě parametru příslušný obrázek poskytnul (pomocí vstupního proudu jej načetl a pomocí výstupního proudu jej zapsal do výstupu servletu). V případě chybějícího obrázku, by servletu poskytnul Spring Framework třídu, která obrázek vygeneruje a uloží. Musí se ovšem počítat s faktem, že servlet existuje v jediné instanci a požadavky jsou vyřizovány pomocí nových vláken. Je zapotřebí ošetřit situace, kdy se jeden klient bude dotazovat na náhled, který je zrovna generován (v praxi to znamená synchronizovat příslušné metody).

Jak zapsat obrázek do výstupu servletu ukazuje následující kód.

```
response.setContentType("image/png");
InputStream in = new BufferedInputStream(new
FileInputStream(file));
OutputStream output = response.getOutputStream();
byte[] b = new byte[1024];
int bytesRead = 0;
while ((bytesRead = in.read(b)) >= 0) {
output.write(b, 0, bytesRead);
```

Klient může náhled získat v podobě objektu *BufferedImage* při použití třídy *ImageIO* a její metody *read(InputStream input)*. Příslušný vstupní proud lze získat vytvořením *URLConnection* a zavoláním metody *getInputStream()*, jak je ukázáno níže.

```
public static BufferedImage getImage(String urlArgs) throws
MalformedURLException,IOException{
```

```
    SearchClient client = SearchClient.getInstance();
    URL url = new URL(client.getWebServiceUrl() + urlArgs);
    URLConnection con = url.openConnection();
    BufferedImage img = ImageIO.read(new
BufferedInputStream(con.getInputStream()));
    return img;
}
```

3.3 Ukládání uživatelských nastavení

Jedním z dalších požadavků bylo zachování uživatelských nastavení i po restartu aplikace. Při pročítání dokumentace jsem zjistil, že na úrovni platformy existuje mechanismus, který je pro tento účel ideální.

Platforma umožňuje ukládání informací do souborů *.properties*, ať už pro jednotlivé moduly nebo pro celou aplikaci. Soubory jsou uloženy při ukončování aplikace do domovského adresáře a roztřízeny do složek jednotlivých balíčků (v případě, že se jedná o soubory jednotlivých modulů). Při startu jsou tyto soubory opět načteny a můžeme k nim ve zdrojovém kódu přistupovat.

V balíčku *org.openide.utils* se nachází třída *NbPreferences*. Tato třída má dvě statické metody, které mají návratový typ *java.util.prefs.Preferences*:

- **root()** – vrací objekt *Preferences* pro celou aplikaci
- **forModule(Class cls)** – vrací *Preferences* pro daný modul (na základě objektu *Class*)

Preferences funguje na principu Key-Value (podobně jako *java.util.Map*). Struktura *.properties* souboru je následující:

```
DESC_UMR=UMR Universal Map Repository
VISIBLE_FABS=UMR,CZ4,PHD,MYD,USR,USC
USER_AUTH=ONDEX
IMAGE_LOADER_PATTERN=/ImageLoader?id={0}&fileName={1}
SAVE_LOTID_AS=Promis Lot Id,Trak Lot Id,Free Lot Id,Foundry Lot
Id
DEFAULT_LOTID=Promis Lot Id
```

Preferences obsahuje metody pro získávání hodnot na základě klíče, přidávání nebo mazání jednotlivých záznamů. Můžeme si nechat vrátit pole se všemi klíči, případně hodnotami. Práce je velice jednoduchá a intuitivní.

Třída *NbPreferences* je ovšem dostupná pouze na úrovni platformy. Vzhledem k tomu, že v aplikaci se používají komponenty i z obyčejných JAR souborů, které následně aplikace využívá, nastává menší komplikace. Je zapotřebí vyřešit otázku jak

z těchto komponent ukládat data na úrovni platformy neboli jak volat metody na vyšší úrovni z úrovně nižší. Ať se jeví tento problém jako těžce řešitelný, jeho řešení je vskutku jednoduché – stačí implementovat rozhraní. Na úrovni knihovny si deklarujeme rozhraní, které bude poskytovat metody pro ukládání a načítání dat. Tyto metody bude volat komponenta na nižší úrovni. Na úrovni platformy implementujeme příslušné rozhraní a dopíšeme obsah přepsaných metod pro ukládání a načítání nastavení. Aby tento mechanismus fungoval, musíme při startu aplikace registrovat třídu na úrovni platformy jako posluchače (listener) v komponentě na nižší úrovni. To lze snadno docílit přidáním příslušného parametru do konstruktoru komponenty.

Implementace tohoto mechanismu do aplikace ovšem nutí programátora k zásadním změnám. Vzhledem k tomu, že se takto načítají jistá nastavení a data, není vhodné uchovávat nějaký výchozí stav v programovém kódu, který bude v případě uložení uživatelských nastavení „přepsán“. Museli bychom mít v aplikaci dva mechanismy:

- **První mechanismus** by nastavil aplikaci dle nějakého výchozího schématu ve zdrojovém kódu aplikace.
- **Druhý mechanismus** by toto nastavení následně změnil z dostupných údajů v *.properties souborech.

Tento nesoulad jsem vyřešil podsunutím výchozího konfiguračního souboru při startu aplikace v případě, že dosud žádný konfigurační soubor (nebo soubory) neexistuje. Ve zdrojovém kódu se již aplikace nastavuje výhradně z údajů v *.properties* souborech. Různých způsobů nastavení lze již docílit pouze deklarativním způsobem. Kde se takový výchozí soubor nachází a jak jej načíst při startu aplikace je popsán v následující kapitole.

3.4 Modul toolsuiteinit

Modul toolsuiteinit byl vytvořen speciálně pro inicializaci uživatelských nastavení a přihlašování do aplikace. Je to jediný modul, který obsahuje třídu Installer (potomek třídy *org.openide.modules.ModuleInstall*), takže je to také jediný modul, který vykonává nějaký kód při startu aplikace.

Při startu se kontroluje existence uložených nastavení. V případě, že nic není nalezeno, se načtou výchozí hodnoty ze souboru, který je uložen ve stejném balíčku, a ty se následně zapíšou do *NbPreferences*, jak ukazuje následující kód.

```
input =
this.getClass().getResourceAsStream("/com/onsemi/cim/apps/toolsu
itebe/nb/toolsuiteinit/Preferences.properties");
reader = new BufferedReader(new InputStreamReader(input));
String line;

while ((line = reader.readLine()) != null) {
    if (!line.startsWith("#")) {
        pref.put(line.substring(0, line.indexOf("=")),
line.substring(line.indexOf("=") + 1, line.length()));
    }
}
```

Ve zdrojovém kódu se od této chvíle můžeme vždy bez obav odkazovat výhradně na *NbPreferences*.

Další funkcí tohoto modulu je zobrazení přihlašovacího okna při startu. Uživatel do něj zadá své údaje a v případě jeho verifikování start pokračuje. V opačném případě se aplikace okamžitě ukončí. V případě schválení uživatele se jeho údaje a práva uloží do singletonu *UserInformation*, který se nachází v modulu *databeans* (je všem ostatním modulům dostupný).

3.5 Implementace uživatelských práv a přihlašování do aplikace

Pro zvýšení zabezpečení aplikace bylo rozhodnuto o implementaci přihlašovacího systému. Tento systém by autentizoval uživatele při spuštění aplikace a v případě potřeby i během jejího používání.

3.5.1 Klientská část

V klientské aplikaci je zapotřebí ihned po startu aplikace zobrazit přihlašovací dialog. Tento dialog zobrazíme pomocí přímo určeného API platformy. Níže uvedený kód ukazuje konkrétní použití API plus mechanismus pro maximální počet pokusů pro přihlášení. Kód jsem umístil do třídy *Installer* modulu *toolsuiteinit*, aby bylo zaručeno zobrazení dialogu co nejdříve. Je ovšem nutné brát na vědomí, že start aplikace se o tuto proceduru zpomalí.

```
SearchClient client = SearchClient.getInstance();

        if (!client.isPasswordValid(panel.getUserName(),
panel.getPassword(), serviceType)) {
            if (attemptsCount >= 2) {
                NotifyDescriptor desc = new
NotifyDescriptor.Message("Maximum attempts reached!",
JOptionPane.ERROR_MESSAGE);
```

```
                DialogDisplayer.getDefault().notify(desc);
                attemptsCount = 0;
                d.setClosingOptions(null);
            } else {
                NotifyDescriptor desc = new
NotifyDescriptor.Message("Wrong user name or password!",
JOptionPane.ERROR_MESSAGE);
                DialogDisplayer.getDefault().notify(desc);
                attemptsCount++;
            }
        } else {

UserInformation.getDefault().setUserInfo(client.getUserInfo(pane
l.getUserName(), serviceType));
            d.setClosingOptions(null);
            attemptsCount = 0;
        }
```

Bylo vhodné přidat do nástrojové lišty aplikace panel, který by zobrazoval aktuálně přihlášeného uživatele a umožňoval přihlášení dalšího uživatele. Vytvoření takového panelu je zcela triviální. Obsahuje *JLabel*, který zobrazuje aktuálního uživatele, a tlačítko, které zobrazí příslušný přihlašovací dialog. Abychom dostali tento panel do nástrojové lišty, je zapotřebí trochu vynalézavosti. Vytvoříme si třídu, která implementuje rozhraní *org.openide.util.actions.Presenter.Toolbar*. V rámci této třídy přepíšeme metodu *getToolbarPresenter()* v které vrátíme náš vytvořený panel. Nyní se již můžeme na tuto třídu odkázat v souboru layer.xml, ve kterém definujeme umístění na hlavním panelu.

```
public final class UserSiteInfo implements Presenter.Toolbar,
RevalidateUserCallback {
    private SiteUserPanel panel;

    public UserSiteInfo() {
        super();
        panel = new SiteUserPanel();

panel.setSiteLabelText(SearchClient.getInstance().getWebServiceU
rl());
        init();
    }

    @Override
    public Component getToolbarPresenter() {
        return panel;
```

```
    }
}
```

3.5.2 Serverová část

Programování serverové části měl na starost jiný člen vývojového týmu, avšak shrnu nejpodstatnější funkcionalitu. Přes webové služby se klient dotáže, zdali uživatel existuje a souhlasí heslo. Existují dva způsoby ověření. První způsob využívá adresářovou službu Active Directory. Tento způsob má výhodu v ověřování aktuálně přihlášených uživatelů na konkrétní stanici v rámci doménové sítě. Druhý způsob funguje přes interní systém společnosti, který slouží k přihlášení do některých vnitropodnikových služeb.

Při úspěšném ověření uživatele se systém dotáže na jeho uživatelská práva v rámci aplikace – právo na upload nových map a právo na mazání map. Opět se využívá Spring Framework pro správu příslušných Java Beans a pro jejich deklarativní nastavení, včetně příslušných SQL dotazů (viz ukázka níže).

```
<bean id="User"
class="com.onsemi.cim.apps.toolsuite.be.user.User" >
    <property name="dataSource" ref="MSDataSource" />
    <property name="sqlSelectUserRight">
        <value>
                SELECT r.*
                  FROM ms_right r, ms_user_right ur, ms_user u
                 WHERE r.id = ur.id_right
                   AND u.id = ur.id_user
                   AND UPPER(u.user_id) = UPPER(?)
        </value>
    </property>
</bean>
```

3.6 Callback

Aby se některé TopComponenty v aplikaci aktualizovaly (v případě, že naslouchají změnám lookupu), je zapotřebí si pro ně vyžádat fokus (nastavit je jako aktivní). To může být v některých případech problematické. Uživatelé aplikace si přáli možnost používání klávesnice pro procházení tabulky v modulu YieldTable. Aby mohly být použity kurzorové šipky na klávesnici k ovládání nějaké komponenty, je nezbytné mít na této komponentě neustále fokus. Pokud ovšem budeme mít neustále fokus na komponentě YieldTable, nemusí se správně aktualizovat ostatní komponenty. Navíc je třeba mít na vědomí, že se jednotlivé moduly navzájem nevidí.

Problém jsem vyřešil vytvořením nového rozhraní *YieldComponentCallback*, které obsahuje jedinou metodu – *callback(String component)*. Instanci třídy implementující toto rozhraní vložíme do lookupu při změně kurzoru v tabulce v komponentě YieldTable. U všech modulů naslouchajícím změnám v YieldTable registrujeme posluchače následujícím způsobem:

```
Lookup.Template<YieldComponentCallback> callbackTemplate = new
Lookup.Template<YieldComponentCallback>(YieldComponentCallback.c
lass);
            callbackResult =
Utilities.actionsGlobalContext().lookup(callbackTemplate);
            callbackResult.addLookupListener(new
LookupListener() {

                @Override
                public void resultChanged(LookupEvent
lookupEvent) {
                    Lookup.Result r = (Lookup.Result)
lookupEvent.getSource();
                    Collection c = r.allInstances();
                    if (!c.isEmpty()) {
                        callback = (YieldComponentCallback)
c.iterator().next();
                    }
                }
            });
```

Po zpracování dat zavolá každý modul metodu *callback(String component)*, kde jako parametr uvede název své třídy.

```
if (callback != null && file != null) {
            callback.callback(this.getClass().getName());
        }
```

V tuto chvíli budou jednotlivé moduly po své aktualizaci volat metodu *callback* rozhraní *YieldComponentCallback*. V místě implementace tohoto rozhraní již stačí dopsat pouze tělo metody callback tak, aby se po zavolání všemi patřičnými moduly přepnul fokus zpět na YieldTable. Musí se ovšem myslet na fakt, že nemůžeme čekat odpověď od komponent, které v tu chvíli budou zavřené.

Ve zdrojovém kódu níže je ukázáno, jak zjistit množinu otevřených TopComponent, získat názvy tříd jednotlivých instancí a porovnat je s názvem třídy v parametru metody. Tímto mechanismem zjistíme, která komponenta je již aktualizovaná a která ne. Pokud jsou všechny otevřené komponenty aktualizované,

zavoláme metodu, která obstará získání fokusu. Nakonec se hodnoty všech proměnných resetují, aby bylo umožněno jejich opětovné použití.

```
callback = new YieldComponentCallback() {
            @Override
            public void callback(String component) {
                Set<TopComponent> tset =
WindowManager.getDefault().getRegistry().getOpened();
                Set<String> set = new HashSet<String>();
                for (TopComponent t : tset) {
                    set.add(t.getClass().getName());
                }
                if
(component.equals("com.onsemi.cim.apps.toolsuitebe.typepareto.Ty
peParetoTopComponent") ||
!set.contains("com.onsemi.cim.apps.toolsuitebe.typepareto.TypePa
retoTopComponent")) {
                    typePareto = true;
                }
                if (typePareto{
                    typePareto = false;
                    activate();
                    content.remove(this);
                }
                tset = null;
                set = null;
            }
        };
```

Závěr

V rámci bakalářské práce jsem se zaměřil především na popsání principu fungování platformy NetBeans. Ovšem platforma NetBeans není jedinou moderní technologií a ve spojení s ostatními dokáže tvořit velice robustní základ pro opravdu rozsáhlé a náročné aplikace. Tyto teoretické znalosti jsem aplikoval na vývoj programu pro výrobu polovodičových součástek ve společnosti ON Semiconductor. V práci jsem se věnoval i nástrojům, které přímo nesouvisí s platformou NetBeans či ostatními frameworky, ale které jsou pro pohodlný vývoj naprosto zásadní. Mám na mysli nástroje Apache Ant a Apache Maven. Bez těchto nástrojů by byl vývoj rozsáhlého systému, členěného do mnoha podprojektů, naprosto nepředstavitelný.

Nezaměřil jsem se výhradně na IT stránku problematiky vývoje aplikací. Snažil jsem se i představit finanční úhel pohledu a také do malé míry problematiku lidských zdrojů. Nelze přeci vyvíjet software bez peněz a bez lidí.

Jako cíl mé práce jsem si vytyčil představit moderní technologie, analyzovat prostředí a prakticky ukázat vývoj na těchto technologiích postavený. Cíl se mi podařilo splnit pouze do jisté míry. Moderních technologií je nepřeberné množství a jejich představení a popis není možné sloučit do rozsahu bakalářské práce, aby měla jistou vypovídací hodnotu. V rámci možností jsem se zaměřil na konkrétní aplikaci a popsal technologie, které využívá.

Během své letní stáže ve společnosti ON Semiconductor, jsem ve svých znalostech postoupil kupředu o mílové kroky. Zjistil jsem, co všechno obnáší programování velkých aplikací a také jsem pochopil jak je nezbytný dobrý objektový návrh aplikace, což při malých projektech člověk nepochopí. To vše jsou vědomosti, které dle mého názoru škola člověku předat nedokáže. Celá práce měla velice pozitivní dopad na vývoj pracovních vztahů mezi mou osobou a společností.

Seznam použité literatury

Monografické publikace:

1) BÖCK, H. *Platforma NetBeans - Podrobný průvodce programátora.* 1. vyd. Brno: Computer Press, 2010. 320 s. ISBN 978-80-251-3116-9.

2) CHING, M. O. – PORTER, B. *Apache Maven 2 Effective Implementation.* 1st ed. Birmingham: Packt Publishing, 2009. 456 s. ISBN 978-1-847194-54-1.

3) PETRI, J. *NetBeans Platform 6.9 Developers's Guide.* 1st ed. Birmingham: Packt Publishing, 2010. 288 s. ISBN 978-1-849511-76-6.

4) ZAHKOUR, S., et al. *Java 6 Výukový kurz.* 1. vydání. Brno: Computer Press, 2007. 536 s. ISBN 978-80-251-1575-6.

Internetové zdroje:

5) *Introduction to Spring Framework.* [Online]. [cit. 2011-04-20]. Dostupné z WWW: <http://static.springsource.org/spring/docs/3.1.0.M1/spring-framework-reference/html/overview.html>.

6) *The Apache Ant Project* [Online]. 2011 [cit. 2011-04-04]. Dostupné z WWW: <http://ant.apache.org/>.

7) BRANICKÝ, M. Java Servlets - predstavenie technológie. *Interval.cz* [Online]. 2003 [cit. 2011-04-24]. Dostupný z WWW: <http://interval.cz/clanky/java-servlets-predstavenie-technologie/>.

8) HYNAR, M. ANT - Nebojte se mravence. *Root.cz* [Online]. 2003 [cit. 2011-04-04]. Dostupný z WWW: <http://www.root.cz/clanky/ant-nebojte-se-mravence/>.

9) CHURÝ, L. Základy XML webových služeb. *Programujte.com* [Online]. 2005 [cit. 2010-10-15]. Dostupný z WWW: <http://programujte.com/?akce=clanek&cl=2005081704-zaklady-xml-webovych-sluzeb>.

10) *JFreeChart* [Online]. 2003 [cit. 2011-04-09]. Dostupný z WWW: <http://www.jfree.org/jfreechart/>.

11) Lesson: Concurrency in Swing. *The Java™ Tutorials* [Online]. 2011 [cit. 2011-03-15]. Dostupný z WWW: <http://download.oracle.com/javase/tutorial/uiswing/concurrency/index.html>.

12) Maven - Available Plugins. *Apache Maven Project* [Online]. 2011 [cit. 2011-04-09]. Dostupný z WWW: <http://maven.apache.org/plugins/>.

13) Maven - What is Maven? *Apache Maven Project* [Online]. 2011 [cit. 2011-04-09]. Dostupný z WWW: <http://maven.apache.org/what-is-maven.html>.

14) NetBeans IDE 7.0 Release Information [Online]. 2011 [cit. 2011-04-09]. Dostpný z WWW: <http://netbeans.org/community/releases/70/>.

15) NetBeans IDE - NetBeans Rich-Client Platform Development (RCP) [Online]. 2010 [cit. 2010-10-11]. Dostupný z WWW: <http://netbeans.org/features/platform/index.html>.

16) Overview of Enterprise Applications - Your First Cup: An Introduction to the Java EE Platform [Online]. 2011 [cit. 2011-04-20]. Dostpný z WWW: <http://download.oracle.com/javaee/6/firstcup/doc/gcrky.html>.

17) SwingLabs.org [Online]. 2010 [cit. 2011-04-09]. Dostpný z WWW: <http://swinglabs.org/projects.jsp>.

18) SwingWorker (Java Platform SE 6) [Online]. 2011 [cit.2011-03-15]. Dostupný z WWW: <http://download.oracle.com/javase/6/docs/api/javax/swing/SwingWorker.html>.

19) Worker Threads and SwingWorker [Online]. 2011 [cit.2011-03-15]. Dostupný z WWW: <http://download.oracle.com/javase/tutorial/uiswing/concurrency/worker.html>.

Seznam zkratek

API	Application Programming Interface
apod.	a podobně
atd.	a tak dále
CPU	Central Processing Unit
DB	databáze
DBMS	Database Management Systém
EIS	enterprise information system
ERP	Enterprise Resource Planning
GUI	Graphical User Interface
HTTP	hypertext transfer protokol
IDE	Integrated Development Enviroment
IS	informační systém
IT	informační technologie
JAR	Java Archive
Java EE	Java Enterprise Editon
JDBC	Java Database Conectivity
JVM	Java Virtual Machine
JAX-WS	Java Api for XML Web Services
JSF	Java Server Faces
JSP	Java Server Pages
např.	například
NBM	NetBeans Module
ORM	Object-relational Mapping
OS	operační systém
PC	Personal Computer
PDF	Portable Document Format
POJO tg	Plain Old Java Object
RAM	Random Access Memory
RCP	Rich-Client Platform
SQL	Structured Query Language
tzv.	takzvaně
URL	Uniform Resource Lokator

WAR	Web Application Archive
WSDL	Web Service Definiton Language
XML	Extensible Markup Language

Prohlášení o využití výsledků bakalářské práce

Prohlašuji, že

- jsem byl seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, bakalářskou práci užít (§ 35 odst. 3);
- souhlasím s tím, že bakalářská práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO a jeden výtisk bude uložen u vedoucího bakalářské práce. Souhlasím s tím, že bibliografické údaje o bakalářské práci budou zveřejněny v informačním systému VŠB-TUO;
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;
- bylo sjednáno, že užít své dílo, bakalářskou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne 9. května 2011

……
Miroslav Zapletal

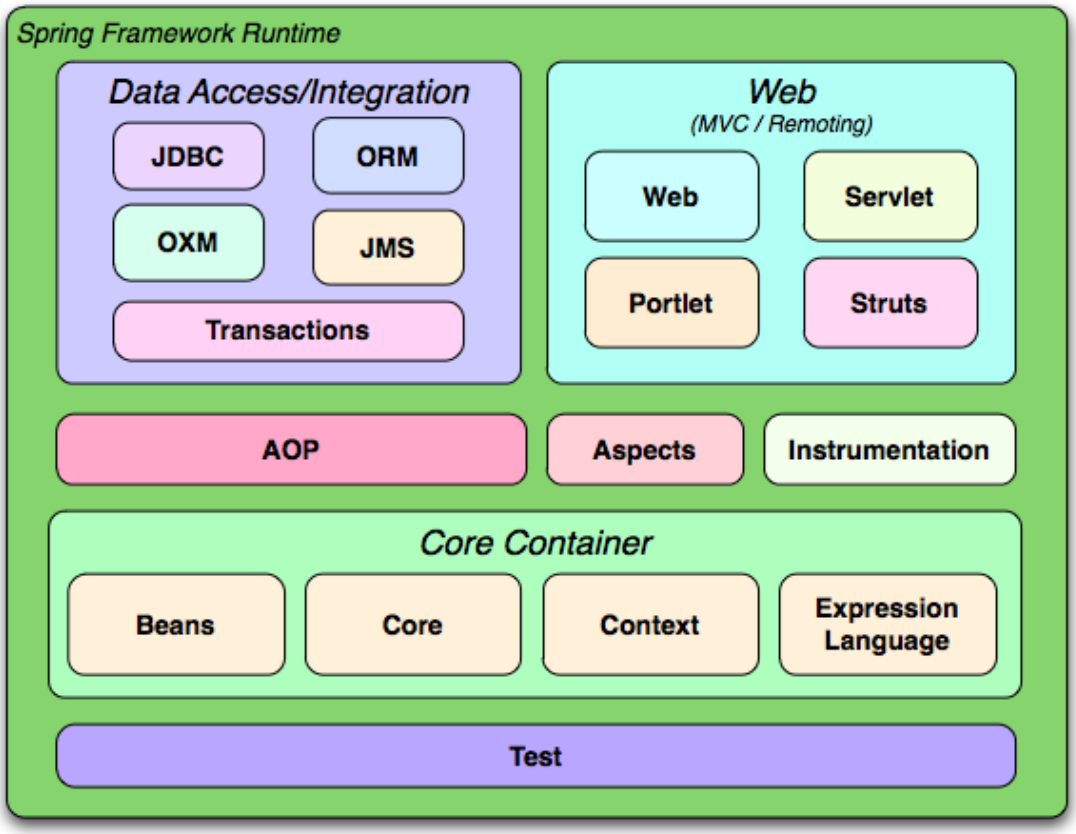
Adresa trvalého pobytu studenta:
Sluneční 2372, 756 61 Rožnov pod Radhoštěm

Seznam příloh

Přílohy

Příloha č. 1: Schéma Spring Frameworku

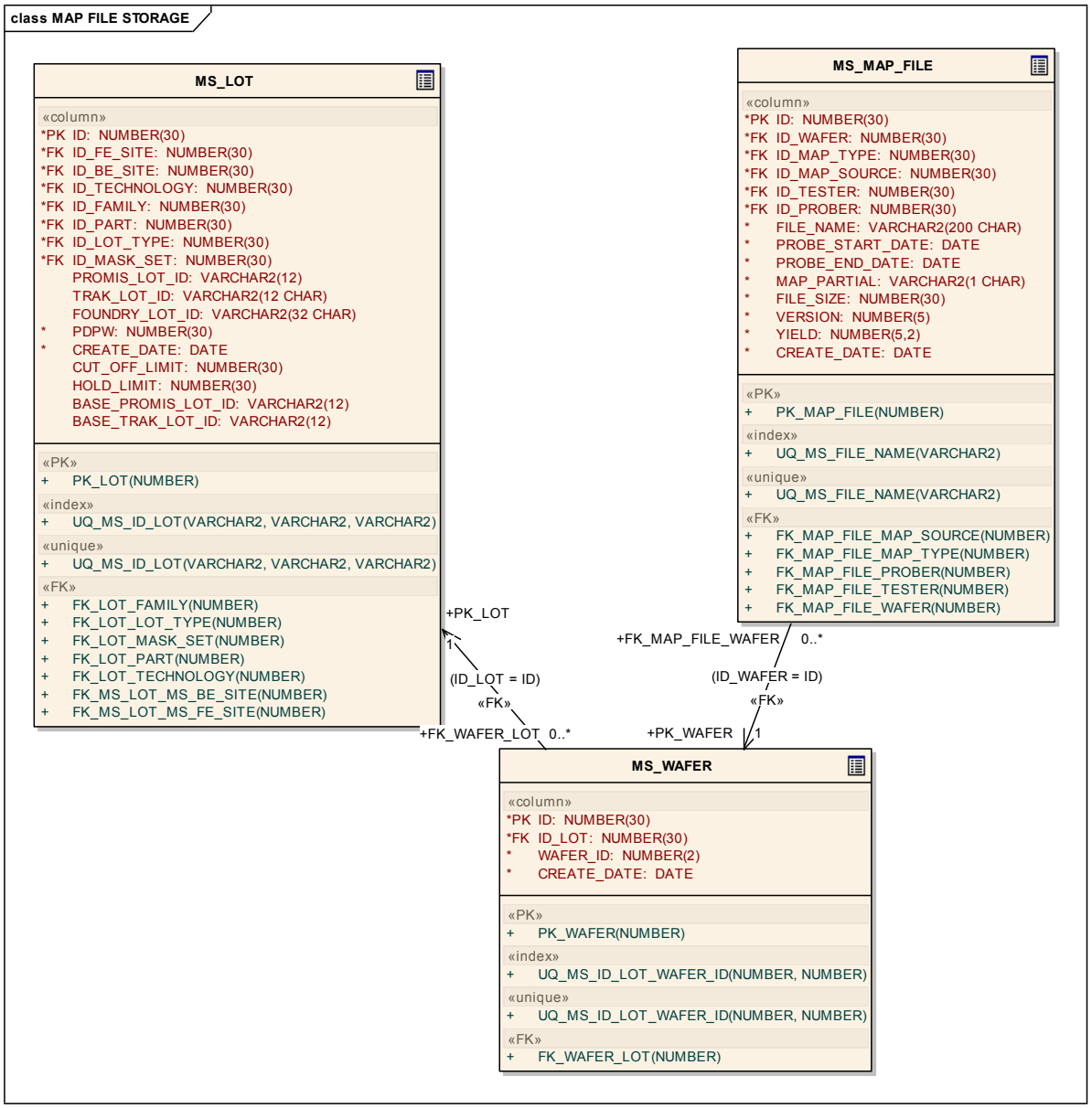
Obrázek 1.5: Schéma Spring Frameworku



Zdroj: [19]

Příloha č. 2: Databázové schéma

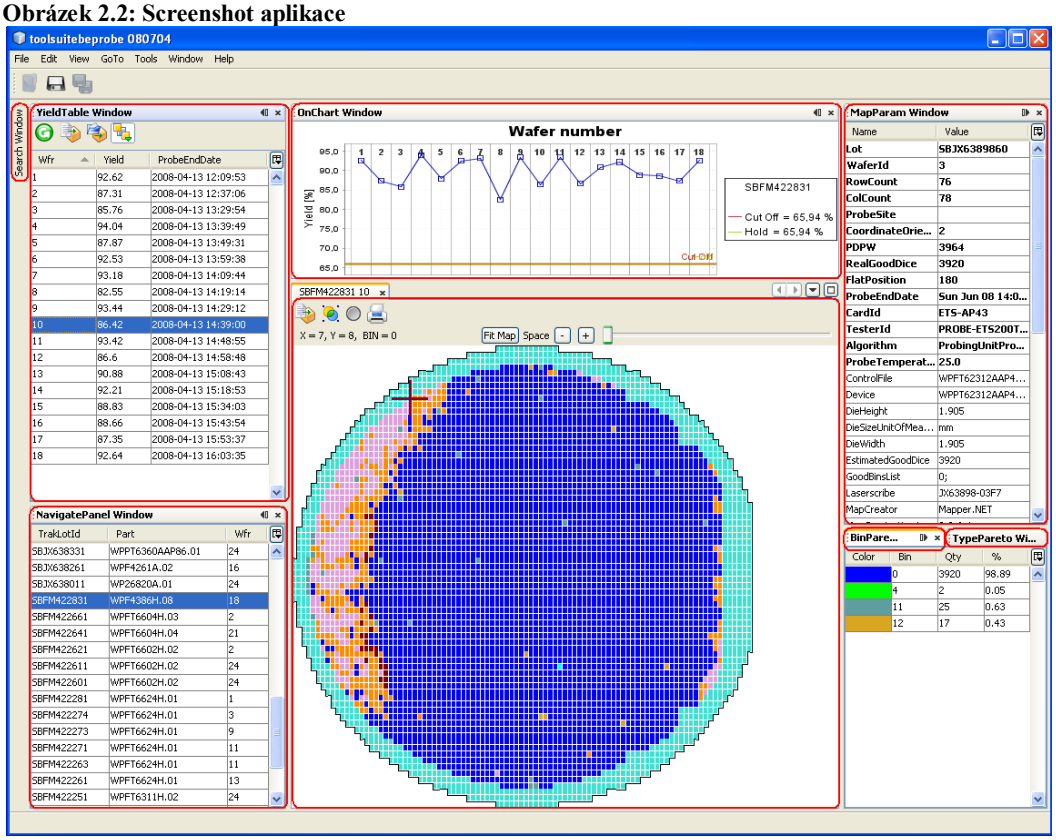
Obrázek 2.1: Databázové schéma



Zdroj: [Interní zdroj]

Příloha č. 3: Screenshot aplikace

Obrázek 2.2: Screenshot aplikace



Zdroj: [Interní zdroj]